

Nr. 11

Best.-Nr. 38 745

BASIC-BOSS

Dieser Basic-Compiler macht Ihre Programme bis zu 100mal schneller

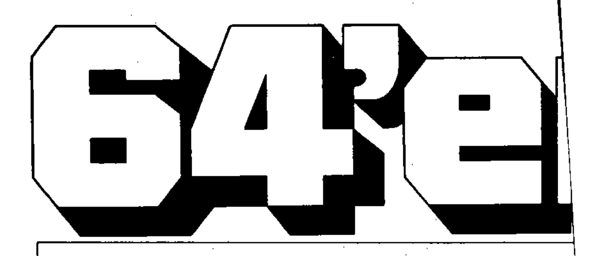




15 14 13 12 11 10 9 8 7 6 5 4 3 2
91 90 89

Bestell-Nr. 38745

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Bosch-Druck, Landshut
Printed in Germany



BASIC-BO
Basic-Comp

Eine 5¼″-Diskett



Markt&Technik Verlag Aktiengesellschaft · Hans-Pinsel-Stra

Die Informat

Bei de

Verlag, He

Für

Commo
»Commodo
Commodore
Der

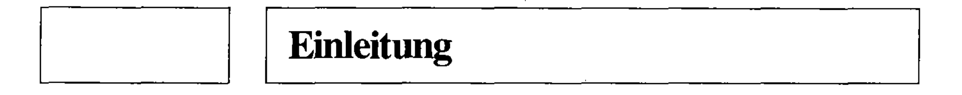
Inhaltsverzeichnis



Vorwort

Die meisten hatten nicht mehr damit gerechnet, daß noch ein Basic-Compiler für den C64 erscheint, der das bisher Dagewesene in den Schatten stellt. Nun ist es aber doch noch Wirklichkeit geworden: Basic-Boss übersetzt Ihre Programme in *reine* Maschinensprache. Versehen Sie Ihre Programme noch mit sogenannten »Compilerdirektiven«, so können Sie Ihre Programme noch weiter beschleunigen. Dabei sind Beschleunigungen um den Faktor »100« keine Seltenheit. Sie können in Zukunft in vielen Fällen auf die aufwendige Programmierung in Maschinensprache verzichten.
Viele Features erlauben ein komfortables und leistungsgerechtes Arbeiten, denn Basic-Boss enthält unter anderem neue Befehle und erlaubt das Kompilieren von Programmen, die mit diversen Basic-Erweiterungen erstellt wurden. Außerdem stellt der Basic-Boss dem Programmierer eine Parametertabelle bereit, um häufig gebrauchte Grundeinstellungen abzuspeichern.
Wir wünschen Ihnen viel Erfolg mit dem Basic-Boss.
Ihre Software-Extra-Redaktion





Einleitung

Einst hatte ich die Idee, ein Basic-Programm direkt in Maschinensprache umzuwandeln. Da das Ganze aber etwas eintönig war, wollte ich den Prozeß automatisieren und setzte mich vor meinen C64 und programmierte einen Basic-Compiler. Nach über einem Jahr harter Arbeit und vieler Entbehrungen war es dann soweit: Der Basic-Boss erblickte das Licht der Welt. Er (Quellcode) erstreckt sich inzwischen über zwei Diskettenseiten (jeweils 41 Tracks) und ein vollständiger Assemblerlauf benötigt zirka 30 Minuten. Die Assemblierung ist allerdings nur auf einer 512-Kbyte-RAM-Disk möglich (ich benutze Turbo Trans, welches jedoch einige nicht unwesentliche Nachteile besitzt. So löscht es zum Beispiel hin und wieder unmotiviert einzelne Sektoren oder ganze Diskettenseiten). Der Assembler, der ein solches Riesenprojekt überhaupt noch möglich macht, ist der »Assi« von Dirk Zabel. Einen annähernd gleichwertigen Assembler konnte ich übrigens bislang nicht einmal auf den »Proficomputern« Amiga und Atari ST ausmachen.
Ein solch umfangreicher Compiler benötigt auch eine ausführliche Anleitung. Diese besteht aus zwei Kapiteln. Kapitel »1« vermittelt Grundlagen und führt in die Bedienung des Basic-Boss ein. Die meisten von Ihnen werden weite Teile davon überspringen können, da dort sicherheitshalber ganz von vorne angefangen wird und der Anwender fast ohne Grundwissen auskommt. Dieses Kapitel erhebt aber keinen Anspruch darauf, genau oder gar vollständig zu sein, sondern ist als Einführung gedacht. Trotzdem kann man nach dessen Lektüre bereits mit dem Basic-Boss arbeiten. Im Gegensatz zu Kapitel »1« bietet Kapitel »2« eine genaue und vollständige Beschreibung aller Befehle und Eigenarten des Basic-Boss. Aber auch dort wird meist nicht viel Grundwissen vorausgesetzt.
Um das Handbuch verstehen und den Compiler benutzen zu können sind allerdings gewisse Grundkenntnisse in der Programmiersprache Basic nötig, da nur die Eigenschaften und Befehle des Compilers beschrieben werden, nicht aber die des Commodore-Basic. Darum ist dem Basic-Einsteiger ein gutes Basic-Buch zu empfehlen, da das Standard-Commodore-Handbuch ziemlich mager ist.

Die gesamte Anleitung wurde übrigens auf dem C 64 mit VizaWrite verfaßt. Ich ziehe den C 64 bei der Textverarbeitung dem Amiga und dem Atari ST immer noch vor (die stehen nutzlos daneben) und halte die Behauptung für unsinnig, der C 64 sei für vernünftige Textverarbeitung zu klein. Bevor Sie mit dem Compiler arbeiten, sollten Sie sich eine Sicherheitskopie anlegen. Dies stellt kein Problem dar, da der Basic-Boss zum einen keinen Kopierschutz beinhaltet und sich zum anderen sogar selbst kopieren kann. Und das geht so: Stellen Sie sicher, daß Sie eine weitgehend leere und formatierte Diskette verfügbar haben und laden Sie dann den Basic-Boss von der Originaldiskette mit

```
LOAD "BASIC-BOSS",8
```

Wenn der Computer fertiggeladen hat, geben Sie »RUN« ein. Jetzt können Sie Ihre formatierte Diskette in das Laufwerk schieben. Anschließend drücken Sie auf die <*>-Taste und geben einen Namen ein, unter dem der Basic-Boss dann abgespeichert wird, wenn Sie die Eingabe mit <RETURN> beenden. Auf der Originaldiskette befinden sich noch ein paar Beispielprogramme, die Sie sich ansehen sollten. Ansonsten wünsche ich Ihnen viel Spaß beim Lesen der Anleitung und hoffe, daß Sie mit dem Basic-Boss keine allzu großen Schwierigkeiten haben.

Kapitel 1 Einführung

1.1 Grundlagen

1.1.1 Basic und Maschinensprache

Frisch ab Werk bietet der C64/C128 dem Benutzer die Programmiersprache Basic. Dies hat seinen Grund. Denn Basic ist eine leicht erlernbare Sprache und darum jedem Anfänger sehr zu empfehlen. Doch nicht nur Anfänger arbeiten mit Basic. Auch Fortgeschrittene und Profis schreiben ihre Programme lieber in Basic als in irgendeiner anderen Sprache, nicht zuletzt weil Basic über leistungsfähige mathematische Funktionen und eine komfortable Stringverwaltung verfügt und durchaus auch die Entwicklung größerer Programme erlaubt. Damit könnte sich Basic gegen seinen einzigen ernstzunehmenden Konkurrenten problemlos durchsetzen. Dieser Konkurrent ist die Maschinensprache. Sie besitzt keine Stringverwaltung, kann nicht mit Fließkommazahlen rechnen und ist auch sonst äußerst primitiv und extrem umständlich. Darüber hinaus dauert es bei größeren Programmen recht lange, bis man das Geschriebene überhaupt austesten kann. Bei Basic genügt die Eingabe von »RUN«. Doch trotz all dieser Nachteile wird die Maschinensprache häufig benutzt, denn sie besitzt einen entscheidenden Vorteil: Sie ist wesentlich schneller als Basic.

Dieser Unterschied ist im verschiedenen Prinzip der beiden Sprachen begründet. Die Maschinensprache wird vom Computer selbst (der Maschine) interpretiert. Darum ist sie so schnell und auch so primitiv, denn es wäre viel zu aufwendig, einen Computer zu bauen, der eine weniger primitive, höhere Programmiersprache versteht. Da man dem Benutzer eine solch primitive Sprache nicht zumuten kann, schreibt man in dieser Maschinensprache ein Programm, das eine höhere Sprache verarbeitet beziehungsweise interpretiert. Dies ist der Interpreter. Nun kann in dieser höheren Sprache programmiert werden, zum Beispiel in Basic. Deren Befehle werden vom Interpreter erkannt und ausgeführt. Dieser Vorgang benötigt allerdings ziemlich viel Zeit, und eben deshalb ist Basic so langsam.

Es scheint also unmöglich, die Vorteile der Programmierung in Basic zu nutzen, wenn man schnelle Programme schreiben will, denn für die meisten Anwendungen ist Basic unerträglich langsam. Es gibt aber einen Ausweg aus diesem Dilemma: Man kann jemanden die eigenen Basic-Programme in die Maschinensprache übersetzen lassen. Da sich für diesen langweiligen Job kaum jemand finden lassen wird, liegt es nahe, diesen Vorgang zu automatisieren und ihn dem Computer zu überlassen, denn zur Erledigung monotoner Aufgaben ist er schließlich da. Der einzige Haken an der Sache ist das hierzu nötige Übersetzerprogramm (Compiler genannt), das enorm aufwendig und kompliziert ist, denn ganz so monoton ist die Aufgabe eben doch nicht. Darum begnügen sich viele Compiler damit, das Basic-Programm in einen etwas effizienteren Spezialcode umzusetzen, der dann aber wieder interpretiert wird, weshalb nur eine mäßige Beschleunigung erreicht wird.
Der erste echte Compiler war der vor längerer Zeit in der Zeitschrift 64'er veröffentlichte As-Compiler, der tatsächlich echten Maschinencode erzeugte. Allerdings verstand er nur ein sehr eingeschränktes Basic und war darum nur sehr bedingt brauchbar. Der Basic-Boss versteht dagegen alle Basic-Befehle und erzeugt sogar einen noch effizienteren Code. Man sollte sich aber darüber im klaren sein, daß auch der Basic-Boss die Maschinensprache nicht überflüssig macht, denn extrem zeitkritische Programme wird man nach wie vor in Maschinensprache schreiben, da ein Compiler niemals so effiziente Maschinenprogramme erzeugen kann, wie ein Mensch sie schreibt. Doch nun kann man mit Basic zumindest größenordnungsmäßig die Geschwindigkeit der Maschinensprache erreichen.

1.1.2 Datentypen

Die geringe Geschwindigkeit des Basic-Interpreters resultiert nicht allein aus dem Interpreterprinzip. Ein weiterer Grund ist die ausschließliche Verwendung von Fließkommazahlen (zum Beispiel 1.2345 oder 572545.2647254). Solche Zahlen kann der Prozessor des C 64 (das ist der für sämtliche Berechnungen und Operationen zuständige Baustein im C 64) nur mit größter Mühe verarbeiten, weshalb es auch ziemlich langsam geht. Darum liegt es nahe, einfachere Zahlen zu verwenden, die der Prozessor leichter verarbeiten kann. Dies sind ganze Zahlen, also völlig normale Zahlen ohne Nachkommastellen. Wenn man sich Basic-Programme

ansieht, so wird man erkennen, daß in den meisten Fällen auch keine Nachkommastellen nötig sind und man statt Fließkommavariablen Ganzzahlvariablen benützen könnte. Solche Variablen gibt es tatsächlich. Hin und wieder stößt man in Basic-Programmen auf Variablen der Form »A%«. Diese Variablen können nur ganze Zahlen von –32768 bis +32767 annehmen. Man könnte folglich vermuten, daß diese Zahlen schneller verarbeitet werden. Die Anweisung A%=B%+C% müßte also schneller sein als A=B+C. Doch dem ist nicht so. Diese sogenannten Integer-Zahlen sind sogar langsamer, was jedoch nicht an den Zahlen selbst, sondern an der fehlenden effizienten Implementierung im Betriebssystem liegt.
Immerhin besitzt das Commodore-Basic damit bereits drei sogenannte Datentypen oder anders ausgedrückt: Basic bietet drei Möglichkeiten, Daten darzustellen. Dies ist zum einen die Darstellung als Fließkommazahlen, auch Real genannt (zum Beispiel A=1.2345), zum anderen die Darstellung als Integer-Zahlen (A%=10000) und außerdem die Darstellung als Zeichenketten, die Strings genannt werden (zum Beispiel A$="Kuckuck"). Somit sind die drei in Basic verfügbaren Datentypen »Real«, »Integer« und »String«.
Aber zwei dieser Datentypen sind für den Prozessor fast völlig ungeeignet (Real und String) und der dritte nur bedingt (Integer). Denn der Integer-Typ ist häufig unbrauchbar, wenn man mit Speicheradressen arbeitet, also mit PEEK und POKE, denn die Speicherzellen des C64 sind bekanntlich von 0 bis 65535 durchnumeriert und nicht von –32768 bis +32767 (in Kapitel »2« wird erklärt, warum beides eigentlich das gleiche ist). Darum liegt es nahe, weitere Datentypen einzuführen, denen dieser Mangel nicht anhaftet. Diese Typen nenne ich »Word« und »Byte«. Eine Variable vom Typ »Word« kann ganze Zahlen von »0« bis »65535« annehmen und ist damit hervorragend auf den C64 und seinen Prozessor zugeschnitten. Noch besser paßt der Typ Byte zum Prozessor, denn mit genau diesem Typ geht der 6510 (der Prozessor des C64) ausschließlich um und dieser Typ paßt exakt in eine der 65536 Speicherstellen des C64. Eine Variable dieses Typs kann ganze Zahlen von 0 bis 255 annehmen. Wenn diese Typen im Basic-Programm benutzt werden, kann man es in ein sehr schnelles Maschinenprogramm übersetzen.

1.2 Direktiven und Compilerbedienung

Es erhebt sich die Frage, wie man diese neuen Datentypen überhaupt verwenden kann, denn der Basic-Interpreter versteht sie ja nicht, und man muß dem Basic-Boss irgendwie mitteilen, daß man sie verwenden will. Sie haben zum Beispiel folgendes Programm geschrieben:

```
10 for i=1024 to 2023
20 poke i,160: next i
```

Von sämtlichen 65 536 Speicherstellen, die der C64 besitzt, wählt sich das Programm die aus, die zwischen der 1024sten und der 2023sten liegen und füllt sie mit dem Wert 160. Weil aber genau dort der Bildschirmspeicher liegt, wird der Bildschirm langsam von oben nach unten gefüllt (bei einigen C64 füllen sich nur die Textzeilen, doch das soll nicht weiter stören). Da man dies gerne etwas schneller und zackiger hätte, wird man das Programm zum Beispiel mit dem Basic-Boss kompilieren.
Und genau das können Sie jetzt tun. Schalten Sie also Computer, Monitor und Laufwerk ein und tippen Sie dieses kleine Basic-Programm ab. Wenn Sie es mit »RUN« starten, können Sie sich von der nicht übermäßig hohen Geschwindigkeit überzeugen. Sodann sollten Sie es auf eine nicht allzu volle Diskette unter dem Namen »TEST« abspeichern. Anschließend schieben Sie die Basic-Boss-Diskette in das Laufwerk und laden den Basic-Boss mit

```
LOAD "BASIC-BOSS",8
```

```
*** Basic-Boss Compiler ***
    Version 2.40
(C) 1988 by Thilo Herrmann

Quelldatei:
█
```

Bild 1: So meldet sich der Basic-Boss nach dem Start

Nun kann es ohne Floppy-Beschleuniger etwas dauern (weshalb ich einen solchen grundsätzlich anraten würde. Auf der Basic-Boss-Diskette finden Sie den Software-Speeder »Ultraload Plus«, der zwar einen Hardware-Speeder nicht ersetzen kann, aber trotzdem eine große Hilfe darstellt. Bitte beachten Sie dazu Anhang E). Wenn wieder »READY.« erscheint, geben Sie »RUN« ein. Falls nun eine Textseite angezeigt wird, so können Sie diese mit einer Funktionstaste (<F1>, <F3>, <F5> oder <F7>) verschwinden lassen. Nun werden Sie nach der Quelldatei gefragt, worauf Sie »TEST« eingeben, nachdem Sie die Diskette eingelegt haben, auf der sich das Testprogramm befindet. Wenn Sie die Eingabe mit <RETURN> abgeschlossen haben, beginnt der Basic-Boss mit seiner Arbeit. Er liest das Basic-Programm von der Diskette und erzeugt ein Maschinenprogramm, das er unter dem Namen »+TEST« abspeichert (dieser Vorgang heißt »kompilieren«). Falls zwischendurch ein Fehler auftaucht, unterbricht er seine Arbeit und wartet auf die Betätigung der <SHIFT>-Taste. Dann müssen Sie den Fehler verbessern und von vorne anfangen.

```
Datenbereich schreiben........
Variablen anlegen.......................
.........

     ***** pass 2 *****

Unterroutinen linken....................
........................................
....................

Datenbereich schreiben........

routinen       :2049 -5845  , $0801-$16d5
daten          :5845 -5904  , $16d5-$1710
programm       :5904 -11794 , $1710-$2e12
variablen      :11794-17958 , $2e12-$4626
hilfsspeicher  :17958-17994 , $4626-$464a
strings        :17994-40960 , $464a-$a000

  '←' fuer Neustart, 'F1' fuer Reset,
  alles andere fuer Ende: █
```

Bild 2: Ein Programm wurde erfolgreich kompiliert

Das fertige Maschinenprogramm (auch »Compilat« genannt) kann ganz normal mit

```
LOAD "+TEST",8
```

geladen und mit »RUN« gestartet werden. Was Sie nun sehen, ist zwar um einiges schneller als das Basic-Programm, kann Sie aber vermutlich noch nicht so recht davon überzeugen, daß der Basic-Boss sein Geld wert sein soll. Denn wenn Sie jeden beliebigen anderen Compiler verwenden, wird das Ergebnis in etwa genauso schnell sein. Warum? Die Antwort ist einfach: Es wurde wie üblich mit Real-Variablen gerechnet und das braucht nun mal seine Zeit. Da aber eigentlich nur ganze Zahlen zwischen 0 und 65 535 im Programm verwendet werden, wäre der Datentyp Word viel sinnvoller. Doch wie teilt man dies dem Basic-Boss mit? Der Basic-Interpreter unterscheidet seine Datentypen (Real, String, Integer) anhand eines an den zweistelligen Variablennamen angehängten Sonderzeichens. Bei Real-Variablen wird nichts angehängt (zum Beispiel »A«), bei String-Variablen wird ein »$« angehängt (zum Beispiel »A$«) und bei Integer-Variablen wird ein »%« angehängt (zum Beispiel »A%«). Der Basic-Boss macht das im Prinzip genauso. Man kann ihm aber auch auf andere Weise mitteilen, von welchem Typ

eine Variable sein soll. Wenn zum Beispiel die Variable »I« des Beispielprogramms vom Typ Word sein soll, dann muß in der ersten Programmzeile der Befehl »£WORD I« stehen, etwa so:

```
5 £word i
```

Dieser Befehl ist eine Compilerdirektive. Sie teilt dem Compiler mit, wie er was zu machen hat (er hat die Variable »I« als Word-Variable zu betrachten). Eine solche Direktive beginnt grundsätzlich mit dem »£«-Zeichen. Der Compiler würde das Programm so akzeptieren. Der Interpreter aber beschwert sich wenn man »RUN« eingibt. Zu Recht, denn »£WORD« versteht er nicht. Darum ist es sinnvoll, ein »REM« vor den Befehl zu schreiben. Dann akzeptiert der Interpreter das Programm und führt es ordnungsgemäß aus – der Basic-Boss ignoriert die REM-Zeile allerdings. Damit dieses Dilemma behoben werden kann, gibt es eine Möglichkeit, einen REM-Befehl für den Basic-Boss unwirksam zu machen. Man hängt ihm einfach einen Klammeraffen an:

```
5 rem@ £word i
```

Dann ignoriert der Basic-Boss den REM-Befehl und wertet die Zeile wie normal aus. Der Interpreter dagegen erkennt den REM-Befehl und ignoriert darum den Rest. Wenn Sie nun Ihr Testprogramm laden, es um diese Zeile ergänzen, wieder auf Diskette abspeichern und kompilieren, dann werden Sie beim Maschinenprogramm (»+TEST«) eine starke Geschwindigkeitszunahme feststellen (dies ist nun in etwa die Geschwindigkeit von »Petspeed« mit Integer-Variablen).

1.2.1 Weitere Compilerdirektiven

Aber es geht noch schneller. Die FOR-NEXT-Schleife des Basic-Boss ist recht umständlich und langsam gestaltet, weil es leider nicht anders geht, wenn sie genau dasselbe machen soll wie die des Interpreters. Man kann den Basic-Boss jedoch anweisen, es besser zu machen. Dies geschieht mit dem Befehl »£FASTFOR«. Allerdings gibt es dann ein paar Einschränkungen, die Sie sich in Kapitel »2« unter »£FASTFOR« und »£SLOWFOR« ansehen sollten.

Wenn Sie das Testprogramm folgendermaßen umschreiben:

```
5 rem@ £word i: £fastfor
10 for i=1024 to 2023
20 poke i,160: next i
```

und kompilieren, dann läßt sich rein optisch kein Unterschied mehr zur Maschinensprache feststellen. Außer »£FASTFOR« sind besonders die Compilerdirektiven »£ALLRAM« und »£OTHERON« wichtig. »£ALLRAM« weist den Basic-Boss an, den gesamten Speicher des C64 für das Basic-Programm und die Variablen zu benutzen. Wenn »£OTHERON« am Programmanfang steht, dann sind an den SYS-Befehl zum Beispiel per Komma angehängte Parameter erlaubt. Beide Befehle müssen wie üblich am Programmanfang stehen. Für nähere Informationen sollten Sie in Kapitel »2« nachschlagen.

1.2.2 Beispiele

Damit Sie einen Eindruck von der Anwendung der neuen Datentypen und der Compilerdirektiven bekommen, zeige ich hier ein paar kleine Beispiele. Sie werden wie beschrieben kompiliert und ausgeführt.

Beispiel 1:

```
10 rem@ £byte a
20 a=0
30 poke 53280,a: a=a+1
40 if a>15 then a=0
50 goto 30
```

Die Bildschirmrandfarbe wird mit allen 16 Farben durchgeschaltet. Weil die Variable »A« nur Werte von 0 bis 15 annehmen darf, kann sie als Byte-Variable deklariert werden.

Beispiel 2:

```
10 rem@ £byte a(,x: £word #
20 dim a(20)
30 sum=0: mul=1
40 for x=1 to 20
50 input a(x)
60 sum=sum+a(x): mul=mul*a(x): next x
70 print sum, mul
```

In diesem Beispiel werden über »INPUT« 20 Byte-Werte in das Array »A(« eingelesen und deren Summe und Produkt berechnet. Etwas seltsam mutet Zeile 10 an. Hier sieht man, wie Arrays (Felder) definiert werden. Dem Variablennamen wird einfach eine Klammer angehängt. Interessant ist auch der Befehl »£WORD #«. Eine Variable »#« gibt es nicht und kann es nicht geben. Mit diesem Befehl werden vielmehr alle Variablen ohne Zusatz (zum Beispiel »A«, »AB«, »COL« und so weiter nicht aber »A%«, »B$« und so weiter) als Word deklariert, falls sie nicht schon an anderer Stelle mit einem Variablentyp deklariert wurden. Variablen der Form »A%«, »VOL%« und so weiter werden mit »£WORD %« als Word deklariert. Nun sind also »SUM« und »MUL« vom Typ Word und »A(« und »X« vom Typ Byte.

Beispiel 3:

```
10 rem@ £allram: £fastfor: £word i: £byte a(
20 dim a(62000)
30 for i=0 to 62000
40 a(i)=100
50 next i
```

Es wird dem Compiler per »£ALLRAM« mitgeteilt, daß er den gesamten Speicher benutzen soll. Dann wird ein Byte-Feld mit 62000 Elementen dimensioniert und mit 100 aufgefüllt. Nun sind Sie am Ende des ersten Kapitels angelangt. Sie können den Compiler jetzt bedienen und benutzen. Eine genaue Beschreibung des Basic-Boss, seiner Befehle und seiner Eigenschaften finden Sie in Teil »2« des Handbuchs.



Kapitel 2 Bedienungshandbuch

Hier finden Sie eine genaue, vollständige und systematische Beschreibung aller Eigenschaften und Befehle des Basic-Boss. Darum ist dieses Kapitel auch zum Nachschlagen sehr gut geeignet.

2.1 Die Bedienung des Basic-Boss

2.1.1 Der Inhalt der Basic-Boss-Diskette

0	38745 basic-boss	m&t	
193	"basic-boss"	prg	Hauptprogramm
0	"----------------"	del	
9	"lies mich"	prg	Hinweise zur Diskette
0	"----------------"	del	
4	"other"	prg	Beispiel für £OTHERON und £OTHER (Basic)
12	"+other"	prg	Beispiel für £OTHERON und £OTHER (Compilat)
0	"----------------"	del	
1	"newcom.o"	prg	Basic-Erweiterung für !COL (Start mit »SYS 49152«)
3	"newcom.src"	prg	Quellcode zur Basic-Erweiterung (als Textfile)
0	"----------------"	del	
3	"errorread"	prg	ASCII-File-Lader Basic (nur als Compilat lauffähig)
13	"+errorread"	prg	ASCII-File-Lader (Compilat)
0	"----------------"	del	
5	"preferences"	prg	Beispiel für Standardeinstellungen
0	"----------------"	del	
3	"bossdatei"	prg	Demonstriert die neuen Dateibefehle (Basic)

7	"+bossdatei"	prg	Demonstriert die neuen Dateibefehle (Compilat)
0	"----------------"	del	
1	"bossdir"	prg	Zeigt Directory einer Diskette (Basic)
5	"+bossdir"	prg	Zeigt Directory einer Diskette (Compilat)
0	"----------------"	del	
3	"ramrom"	prg	Kopiert den Grafikspeicher ins ROM und zurück (Basic)
6	"+ramrom"	prg	Kopiert den Grafikspeicher ins ROM und zurück (Compilat)
0	"----------------"	del	
3	"ramload"	prg	Lädt ein File ins RAM (Basic)
13	"+ramload"	prg	Lädt ein File ins RAM (Compilat)
0	"----------------"	del	
2	"ramdir"	prg	Quellcode zu »r+ramdir« und »p+ramdir«
5	"r+ramdir"	prg	Demonstriert £ROUTSTART
1	"p+ramdir"	prg	Demonstriert £PROGSTART
0	"----------------"	del	
8	"autoboss"	prg	Demonstriert die Automatikfunktion
3	"auto.short"	prg	Kurzversion zum Einbinden in eigene Programme
0	"----------------"	del	
8	"forreturn"	prg	Demonstriert FOR und RETURN bei falschem Einsatz (Basic)
14	"+forreturn"	prg	Demonstriert FOR und RETURN bei falschem Einsatz (Compilat)
0	"----------------"	del	
7	"fastmuldiv"	prg	Schnelle Multiplikation und Division (Basic)
14	"+fastmuldiv"	prg	Schnelle Multiplikation und Division (Compilat)
0	"----------------"	del	
12	"break"	prg	Demonstriert auf fantastische Weise die Geschwindigkeitszunahme (Basic)
27	"+break"	prg	Demonstriert auf fantastische Weise die Geschwindigkeitszunahme (Compilat)

```
0   "----------------"  del
28  "demo"              prg  Demo zur Bildschirmausgabe (Basic)
38  "+demo"             prg  Demo zur Bildschirmausgabe (Compilat)
0   "----------------"  del
27  "labyrinth"         prg  Spiel (Basic)
59  "+labyrinth"        prg  Spiel (Compilat)
0   "----------------"  del
1   "fastdir"           prg  Schnelle Directory-Anzeige (Basic)
11  "+fastdir"          prg  Schnelle Directory-Anzeige (Compilat)
0   "----------------"  del
1   "bigarray"          prg  Demonstriert £ALLRAM (Basic, nur kompiliert
                             lauffähig)
4   "+bigarray"         prg  Demonstriert £ALLRAM (Compilat)
0   "----------------"  del
37  "wayout"            prg  Spiel
0   "----------------"  del
9   "ultraload plus"    prg  Schnellader
1   "ultraload tool 1"  prg
2   "ultraload tool 2"  prg
0   "----------------"  del
74 blocks free.
```

2.1.2 Bedienung des Compilers

Der Compiler wird ganz normal mit

```
LOAD "BASIC-BOSS",8
```

geladen und mit »RUN« gestartet. Wenn nun eine Textseite auf dem Bildschirm erscheint, können Sie diese mit einer Funktionstaste (<F1>, <F3>, <F5>, <F7>) verschwinden lassen (dies ist aber nicht unbedingt notwendig). Dann erscheint die Copyrightmeldung mit Versionsnummer und der Benutzer wird nach der Quelldatei gefragt. Nun können Sie sich das

Directory mit »$« + <RETURN> anzeigen lassen oder sofort den Filenamen eingeben (bei bloßem Drücken von <RETURN> wird er mit »BASIC« angenommen). Dann startet der Kompiliervorgang. Dieser besteht aus zwei Durchgängen (Pass 1 und Pass 2). Während Pass 1 wird die Quelldatei gelesen und die Verteilung der Adressen durchgeführt. In Pass 2 wird das Basic-Programm noch einmal gelesen, damit daraus das Maschinenprogramm erzeugt werden kann, das gleichzeitig auf die Diskette geschrieben wird. Der Name des erzeugten Maschinenprogrammes wird standardmäßig aus dem der Quelldatei gebildet, indem ihm ein »+« vorangestellt wird. Wenn während des Kompilierens ein Fehler auftritt, dann wird die Nummer der fehlerhaften Zeile angezeigt. Hinter der Nummer ist ein Extrakt des bislang bearbeiteten Teils der aktuellen Zeile zu finden, damit der Fehler leichter lokalisiert werden kann. Dieser Extrakt besteht aus Doppelpunkten, Rechenoperatoren und einem Punkt für jeden Befehl. Darunter steht die eigentliche Fehlermeldung hinter einer Fehlernummer, die das Auffinden des Fehlers im Fehlerverzeichnis (Kapitel 2.12.1) erleichtern soll. Der Compiler hält bei jedem Fehler inne, damit dieser nicht ungesehen wieder vom Bildschirm verschwindet (es sei denn, es wurde der »£PROTOCOL«-Befehl verwendet). Ein Drücken auf die <SHIFT>-Taste beendet diesen Starrezustand. Die Fehlermeldungen werden in beiden Durchgängen angezeigt und erscheinen darum häufig doppelt, was aber nicht weiter stört. Am Ende von Pass 1 meldet der Basic-Boss die nicht dimensionierten Arrays, die dann auf 10 dimensioniert werden. Abbrechen kann man den Kompiliervorgang mit <RUN/STOP> oder zur Not mit <RESTORE>. Nach Beendigung des Kompilierens zeigt der Basic-Boss die Adressen der verschiedenen Bereiche an und fordert den Benutzer auf, nun entweder <←>, <F1> oder irgendeine andere Taste zu drücken. Mit <←> wird der Compiler neu gestartet, <F1> führt einen Reset aus. Jede andere Taste beendet den Basic-Boss ohne Reset. Der Basic-Boss bietet außerdem noch eine Speicherfunktion, mit der man ihn abspeichern kann. Wenn man bei der Eingabe der Quelldatei dem Namen einen Stern vorausstellt (»*«), dann wird der Compiler unter diesem Namen auf Laufwerk »8« geschrieben. Der Klammeraffe kann hier übrigens unbedenklich angewandt werden (zum Beispiel »*@:NEUER BOSS«).

2.1.3 Automatikmodus

Wenn man sehr oft kompiliert, zum Beispiel wenn ein Programm nur als Compilat lauffähig ist, dann wird man sehr oft immer dasselbe eintippen. Es wäre also besser, wenn man die Bedienung des Basic-Boss automatisieren könnte. Eine solche Möglichkeit gibt es in der Tat. Man kann dem Compiler verschiedene Informationen übermitteln. Sie stehen im Kassettenpuffer ab Speicherstelle 900. Wenn er gestartet wird, dann untersucht der Basic-Boss zunächst die Speicherstellen 1000 und 1001. Findet er dort die Werte 123 und 234, so nimmt er an, daß der Benutzer den Automatikmodus wünscht. Aus 1002 holt er sich dann die Gerätenummer für die Quelldatei, aus 1003 die zugehörige Sekundäradresse. In 1004 erwartet er die Länge des Datei-Namens und ab 1005 den Filenamen selbst. Das so beschriebene Programm wird dann kompiliert. Wenn das Kompilieren beendet ist, holt sich der Basic-Boss eine Gerätenummer aus 900, eine Sekundäradresse aus 901, eine Namenlänge aus 902 und einen Namen ab 903. Dann veranlaßt er, daß das so beschriebene Programm nachgeladen und gestartet wird und beendet sich selbst. Das nachgeladene Programm kann zum Beispiel das Compilat sein, das dann sofort ausgetestet werden kann.

Allerdings ist es nicht etwa einfacher, sondern wesentlich umständlicher, dem Basic-Boss auf diese Weise den Namen der Quelldatei mitzuteilen, da es viele POKEs erfordert. Darum sollte man für die POKErei ein kleines Programm einsetzen, das die Speicherstellen ab 900 richtig setzt und anschließend den Basic-Boss lädt. Man kann sich noch mehr Arbeit sparen, wenn man dieses kleine Programm in das zu kompilierende Basic-Programm integriert, es um ein paar Diskettenbefehle und einen SAVE-Befehl erweitert und mit »£IGNORE« und »£USE« klammert, damit es das Compilat nicht sinnlos verlängert. Dann braucht man es nur noch mit »RUN Startzeile« aufzurufen, um das Quellprogramm automatisch abspeichern und kompilieren zu lassen, wobei das Compilat dann ebenfalls automatisch geladen und gestartet wird. Ein Beispielprogramm, das die Vorgehensweise veranschaulicht, ist auf der Programmdiskette unter dem Namen »AUTOBOSS« zu finden. Das Programm basiert auf einer kleinen Basic-Routine, die isoliert und gepackt in »AUTO.SHORT« steht und darum sehr leicht in eigene Programme eingebaut werden kann.

2.2 Allgemeines

2.2.1 Compilerdirektiven

Der Basic-Boss wird fast ausschließlich mittels Compilerdirektiven gesteuert. Dies sind Befehle, die wie andere auch im Basic-Programm stehen. Sie erfüllen aber keine normale Funktion wie »PRINT« oder »POKE«, sondern teilen dem Compiler mit, was er zu tun hat. Alle Compilerdirektiven beginnen mit dem Pfund-Zeichen (£), das auf der C 64/C 128-Tastatur mit einem einzigen Tastendruck (Taste zwischen <-> und <HOME>) erreichbar ist. Dann folgen der Befehlsname und danach eventuell einige Parameter.

2.2.2 Zahlensysteme

Zahlen akzeptiert der Basic-Boss sowohl in dezimaler als auch in hexadezimaler Form, wobei hexadezimale Werte mit einem Dollarzeichen (»$«) eingeleitet werden. Darüber hinaus können sie beliebig lang sein und man braucht auf zufällig entstehende Basic-Befehle keine Rücksicht zu nehmen. So ist zum Beispiel folgendes möglich: »PRINT $F, $DEF, $123456789ABCDEF« oder »GOTO $1E«. Kommazahlen können nur in der dezimalen Form angegeben werden.

2.2.3 Inaktivierung von REM-Befehlen

Der Basic-Boss reagiert bei einem REM-Befehl genauso wie der Interpreter: Er ignoriert den Rest der Zeile. Dies kann man aber verhindern, wenn man direkt hinter den REM-Befehl einen Klammeraffen (@) schreibt. Dann untersucht der Compiler den Zeilenrest als wäre das REM nicht vorhanden. Hinter ein solcherart deaktiviertes REM kann man aufgrund einer Eigenart des Interpreters (Tokenisierung) aber nur Compilerdirektiven und Compilerbefehle schreiben – Basic-Befehle werden dort nicht erkannt. Die Deaktivierung von REM-Befehlen ist nützlich, um Compilerdirektiven vor dem Interpreter zu verbergen, damit sich dieser beim Austesten eines Basic-Programms nicht mit einer Fehlermeldung beschwert, wenn er auf eine Direktive aufläuft.

2.2.4 Das Compilat

Das Compilat wird vom Basic-Boss normalerweise mit einer SYS-Zeile versehen als ganz normales Programm abgespeichert. Es kann dann wie ein Basic-Programm geladen und gestartet werden. Man kann es aber auch in einen bestimmten Speicherbereich legen und aus einem Basic-Programm heraus mit »SYS Startadresse« aufrufen (siehe unter Kapitel 2.4.3 »£ROUTSTART«). Das Basic-Programm arbeitet nach Ablauf des Compilats einwandfrei weiter, da alle für Basic wichtigen Speicherstellen nach Beendigung des Compilats wieder restauriert werden.

2.3 Datentypen

Das System der Datentypen ist wohl der bedeutsamste Aspekt des Basic-Boss. Er verfügt über die Typen Byte, Integer, Word, Real und String. Außerdem gibt es noch den Typ Boolean, der jedoch ziemlich unwichtig ist und in den meisten Fällen wie Byte behandelt wird. Jede Variable im Basic-Programm kann einen dieser Typen annehmen.

2.3.1 Die Wertebereiche der Typen

Real und String sind die in Basic meistbenutzten Datentypen. Ihr Wertebereich dürfte Ihnen bekannt sein. Während man beim Stringtyp schlecht vom Wertebereich sprechen kann, da er Zeichenketten von (fast) beliebiger Länge beschreibt, läßt sich das beim Realtyp besser angeben. Er beschreibt Kommazahlen im Bereich von $+-2.93873588 * 10^{-39}$ bis $+-1.70141183 * 10^{-38}$. Beim Integer-Typ fällt der Wertebereich wesentlich einfacher aus: Er reicht von –32768 bis +32767. Word faßt Werte von 0 bis 65535 ($= 2^{16}-1$) und Byte von 0 bis 255 ($= 2^{8}-1$). Die Typen Byte, Word, Integer und Real werden aufgrund ihres Wertebereiches numerisch genannt. Boolean kann eigentlich nur die Werte »0« und »–1« annehmen. Dieser Typ wird für Entscheidungen benötigt.

2.3.2 Die Deklaration

Der Interpreter erkennt den Datentyp einer Variablen am Zusatz hinter dem Variablennamen. Wenn kein Zusatz folgt, ist für ihn die Variable vom Typ Real (Fließkomma). Wenn »%« folgt, schließt er auf eine Integer-Variable (zum Beispiel »A%«) und bei »$« auf eine String-Variable (zum Beispiel »B$«). Der Basic-Boss hält sich standardmäßig auch an diese Regel. Man kann den Variablen aber auch andere Typen zuweisen. Dies geschieht mit einer Compilerdirektive. Diese sieht allgemein so aus:

£Typ Variablenliste

Anstelle von »Typ« kann jeder der genannten Typen stehen (Boolean, Byte, Integer, Word, Real, String). Die Variablenliste besteht aus einem oder mehreren mittels Komma getrennten Variablennamen. Ein Beispiel:

```
10 £byte a,b%,c
20 £word co
```

Hier werden die Variablen »A«, »B%« und »C« zu Byte-Variablen und die Variable »CO« zur Word-Variable erklärt. Man kann die Variablen aber auch anhand ihrer Zusätze deklarieren. Wenn man zum Beispiel alle Variablen der Form »A%« als Word-Variablen deklarieren will, dann genügt es, statt der möglicherweise sehr vielen Variablennamen in der Deklaration einfach »%« zu benutzen. Konsequenterweise kann man auch die Variablen der Form »A$« anders deklarieren, indem man »$« benutzt. Besonders wichtig ist die Möglichkeit, den Variablen ohne Zusatz einen bestimmten Typ zuzuweisen. Dies ist mit dem Doppelkreuz möglich (»#«). Die Deklaration von Variablen mittels ihres Namens hat allerdings Vorrang vor der Deklaration anhand des Zusatzes. Man kann folglich zum Beispiel sämtliche Variablen mit dem Zusatz »%« als Word definieren und trotzdem der Variable »X%« den Typ Byte zuweisen. Die Reihenfolge der Deklarationen ist dabei grundsätzlich ohne Bedeutung.

Beispiele: `10 £byte #,x%: £word a,b,%,c`

Es werden die Variablen »A«, »B« und »C« und alle mit dem Zusatz »%« als Word deklariert. Allen Variablen ohne Zusatz und der Variable »X%« wird der Typ Byte zugewiesen. Da die

ausdrückliche Deklaration Vorrang hat, ist »X%« vom Typ Byte und nicht vom Typ Word, ebenso wie »A«, »B« und »C« vom Typ Word und nicht vom Typ Byte sind. Die hohe Flexibilität kann man natürlich auch mißbrauchen:

```
10 rem@ £string #,a%: £word $: £real %
20 x$=2: y$=3
30 a="hallo": a%="lollipop"
40 b%=1.23456
50 print mid$(a,x$,y$),a%,b%
```

Der Interpreter steigt mit einem »Type Mismatch« aus, aber der Compiler schluckt das Programm anstandslos.

2.3.3 FAST-Variablen

In der Programmiersprache »C« auf dem Amiga oder dem Atari ST gibt es Registervariablen, die der C-Compiler in den Registern des Hauptprozessors abzulegen versucht, damit sie besonders schnell bearbeitet werden können. Beim Prozessor des C64/C128 wäre dies ein hoffnungsloses Unterfangen, da dieser nur sehr wenige sehr kleine Register besitzt, die er darum andauernd anderweitig benötigt. Aber er besitzt die angenehme Eigenschaft, daß er die untersten 256 Byte seines Adreßbereichs (die sogenannte Zero-Page) besonders effizient bearbeitet. Darum erlaubt der Basic-Boss, ein paar Variablen vom Typ Byte, Word oder Integer (keine Arrays) in diesem Bereich zu plazieren. Bei normalen Operationen bringen solche Variablen einen Geschwindigkeitsgewinn von zirka 25%. Der Speicherplatzverbrauch schrumpft um etwa ein Drittel. Drastischer wirken sich FAST-Variablen bei PEEK und POKE aus. Wenn für die Adresse eine solche Variable statt einer normalen Variable verwendet wird, dann werden diese Operationen um den Faktor drei beschleunigt und benötigen nur ein Drittel des Programmspeichers. Darum sollte man FAST-Variablen vor allem hier anwenden. Bei der Deklaration einer FAST-Variablen wird dem Variablennamen einfach ein Gleichheitszeichen angefügt, dem »FAST« folgt (zum Beispiel »£WORD A=FAST, B=FAST«). Dabei ist allerdings zu beachten, daß nur vier Byte für FAST-Variablen zur Verfügung stehen. Es können also nur zwei Word-Variablen mit FAST deklariert werden oder vier Byte-Variablen oder eine Word-Variable

und zwei Byte-Variablen. Dabei werden die Adressen von 251 bis 254 benutzt. Achtung: Fast-Variablen werden zu Programmbeginn nicht wie alle anderen auf »0« gesetzt. Darum müssen Sie vom Programmierer selbst initialisiert werden.

```
*** Basic-Boss Compiler ***
    Version 2.40
(C) 1988 by Thilo Herrmann

Quelldatei:
test 57

20 ...
62/zuviel 'fast'-variablen
```

Bild 3: Fehlermeldung 62: Es wurden zu viele FAST-Variablen deklariert

2.3.4 Adreßfestlegung bei Variablen

Normalerweise bestimmt der Basic-Boss die Adresse einer jeden Variable selbst. Doch auch Sie können die Adresse einer Variable bestimmen, vorausgesetzt, sie ist numerisch. Dies geschieht ähnlich wie bei der FAST-Deklaration, nur muß anstelle von »FAST« eine Adresse angegeben werden, zum Beispiel so: »£WORD A=49152«. Auch Arrays können so deklariert werden (»£BYTE SCREEN=1024«). Nun können Sie zum Beispiel sehr einfach das Register für die Hintergrundfarbe ansprechen, wenn Sie »COL« so deklarieren: »£BYTE COL=53281«. Außerdem kann man irgendwelche Variablen von Maschinenprogrammen ansprechen, indem man eigene Variablen auf die betroffenen Speicherstellen legt. Ein weiterer Anwendungsbereich ist die Übergabe von Werten an Programme, die nachgeladen werden und die gleichen

Deklarationen beinhalten. Außerdem kann man weitere FAST-Variablen kreieren, indem man normale Variablen auf die Zero-Page-Adressen von $2B bis $2E legt, die vom Compilat sonst nicht angerührt werden. In der Tat ist die Anweisung »£WORD A=$FB« hinsichtlich »A« identisch mit »£WORD A=FAST«. Ebenso wie FAST-Variablen werden an eine bestimmte Adresse gelegte Variablen nicht initialisiert und beinhalten am Programmbeginn irgendeinen zufälligen Wert.

2.3.5 Vor- und Nachteile der Datentypen und Verwendung des richtigen Typs

Man kann sich fragen, wozu denn unbedingt fünf Datentypen nötig sind. Die Antwort ist einfach: Die Verwendung der richtigen Datentypen erhöht die Effizienz eines Programms. Es wird also schneller und kürzer. Wesentlich für die Entscheidung des richtigen Datentyps ist der gewünschte Wertebereich. Außerdem sollte man vorzugsweise immer den Datentyp benutzen, der am wenigsten Speicherplatz verbraucht und am schnellsten ist. Hierfür gilt:

- Byte ist am schnellsten und verbraucht am wenigsten Variablenspeicher (genau ein Byte) und Programmspeicher.
- Word ist etwa halb so schnell und verbraucht genau doppelt so viel Variablenspeicher (zwei Byte) und meist mehr als doppelt so viel Programmspeicher.
- Integer ist bei vielen Operationen genauso schnell wie Word und verbraucht genauso viel Variablenspeicher. Bei Vergleichsoperationen ist Integer aber um einiges langsamer (außer bei »=« und »< >«).
- Real ist *wesentlich* langsamer als die oben beschriebenen Typen und verbraucht ganze fünf Byte Variablenspeicher, aber aufgrund einer speziellen Berechnungsmethode etwas weniger Programmspeicher als Word.
- String ist ein extrem aufwendiger Datentyp und darum auch nicht allzu schnell. Doch die Bearbeitung ist dennoch um einiges schneller als beim Basic-Interpreter, nicht nur was die Garbage-collection anbetrifft. Der Platzbedarf ist mit dem Realtyp vergleichbar. Hinzu kommt noch der Platzbedarf des Stringtextes.

Hieraus geht hervor, daß Sie nach Möglichkeit den Bytetyp benutzen sollten, ansonsten Word, sonst Integer und nur wenn es unbedingt nötig ist Real. Wenn Sie zum Beispiel die Variable »I« ausschließlich zur Stringverarbeitung in der Art »A$=LEFT$(B$, I)« benutzen, dann sollten Sie »I« als Byte-Variable deklarieren, da für »I« nur Werte zwischen »0« und »255« sinnvoll sind. Wenn eine Variable »X« aber in der Form »POKE X,0« benutzt wird, sollte sie als Word definiert sein. Weitere Informationen zur Steigerung der Effizienz von Programmen finden Sie unter »Effiziente Programme (Kapitel 2.8)«. Die bei den Befehlen und Funktionen erwarteten Typen können Sie unter »Typen bei Operationen, Funktionen und Befehlen« (Kapitel 2.3.9) nachlesen.

2.3.6 Umwandlungen von Datentypen

Da die verschiedenen Datentypen miteinander in Konflikt geraten können, müssen sie hin und wieder ineinander umgewandelt werden. Dabei kann man zwei Arten der Umwandlung unterscheiden.

2.3.6.1 Umwandlungen mit Verlust

Sehen Sie sich einmal folgendes Programm an:

```
10 rem@ £real a: £byte b
20 a=97.374546
30 b=a
40 print a,b
```

Unter dem Interpreter läuft dieses Programm einwandfrei. Dort wird ja auch nur mit Real-Variablen gerechnet. Beim Compiler müßte es aber Probleme geben, denn in Zeile 30 wird eine Real-Variable einer Byte-Variable zugewiesen. Dies ist rein technisch eigentlich völlig unmöglich, denn man kann keinen fünf Byte großen und sehr genauen Real-Wert einer primitiven Byte-Variable zuweisen, die doch nur ganze Zahlen zwischen »0« und »255« faßt. Aber der Basic-Boss kompiliert dieses Programm ohne Schwierigkeiten. Wenn man es startet, erkennt man das Prinzip: Es werden einfach sämtliche Nachkommastellen abgeschnitten und aus

97.374546 wird schlicht 97. Doch Real-Zahlen können auch größer sein. Wenn einer Byte-Variable nun zum Beispiel 258.123 zugewiesen wird, reicht das Abschneiden der Nachkommastellen nicht aus. Zusätzlich wird die Zahl dann einfach kleiner gemacht, indem grundsätzlich nur der Rest einer Division durch 256 genommen wird. Einfacher erklärt: Es wird so oft 256 abgezogen, bis das Ergebnis in die Byte-Variable paßt. In diesem Fall würde die Byte-Variable nach der Zuweisung den Wert »2« enthalten (258–256). Ähnlich ist es bei den anderen Umwandlungen. Wenn Real nach Word oder Integer gewandelt werden soll, werden ebenfalls die Nachkommastellen abgeschnitten und nur der Rest der Division durch 65536 übernommen. Dies verwundert etwas bei Integer, da dieser Typ einen ganz anderen Wertebereich hat, doch mehr dazu später. Wenn Word oder Integer nach Byte gewandelt werden, wird wieder nur der Rest einer Division durch 256 übernommen (anders ausgedrückt: Es wird schlicht das Highbyte weggelassen).

2.3.6.2 Umwandlungen ohne Verlust

Bei den bisher beschriebenen Umwandlungen wurde immer etwas verloren beziehungsweise die Genauigkeit nahm ab. Es gibt jedoch noch eine Reihe anderer Umwandlungen, die diesen Nachteil nicht aufweisen. Dies sind Umwandlungen von Byte nach Word, von Byte nach Integer oder von Word nach Real. In diesen Fällen ist der Wertebereich des Quelltyps immer ganz im Wertebereich des Zieltyps enthalten. Die Umwandlung von Boolean ist übrigens ebenfalls verlustlos.

2.3.7 Wertigkeit der Typen und deren Verarbeitung

```
10 rem@ £word w: £real r
20 r=1.5: w=4
30 print r*w
```

Wenn dieses Programm unter dem Interpreter mit »RUN« gestartet wird, dann erscheint das Ergebnis »6« auf dem Bildschirm. Was auch sonst. Bei diesem Beispiel tritt für den Compiler aber wieder ein Typkonflikt auf. In Zeile 30 soll die Real-Variable »R« mit der Word-Variable

»W« multipliziert werden. Der Basic-Boss kann aber keine Variablen von unterschiedlichem Typ verknüpfen. Die Multiplikation kann folglich erst durchgeführt werden, wenn eine der beiden Variablen in den Typ der anderen umgewandelt wurde. Der Compiler kann also zum Beispiel den Inhalt von »R« in den Word-Typ umwandeln, um dann zu multiplizieren. Er würde den Kommawert 1.5 in einen Word-Wert umwandeln und deshalb die Kommastellen abschneiden. Er erhielte dann 1. Dann würde er diese »1« mit »4« multiplizieren. Das Ergebnis wäre »4« und offensichtlich falsch. Er sollte also besser den Inhalt der Word-Variablen »W« in eine Real-Zahl umwandeln. Dann würde der Word-Wert »4« zum Real-Wert »4« und ergäbe mit »1.5« multipliziert das korrekte Ergebnis »6«. Genau dies macht der Basic-Boss. Wenn er die Wahl hat, wandelt er immer den Typ mit dem kleineren Wertebereich in den mit dem größeren um. Dies entscheidet er anhand der Wertigkeit eines Typs. Am geringsten ist die Wertigkeit von Boolean und Byte. Dann folgt Integer, vor Word und vor Real. Bei einem Typkonflikt entscheidet sich der Basic-Boss bevorzugt für den Typ mit der höheren Wertigkeit. Dessen sollten Sie sich immer bewußt sein, wenn Sie einen Typenkonflikt provozieren. Ein weiteres Problem:

```
10 rem@ £word w1,w2: £real r
20 w1=3: w2=2
30 r=w1/w2: print r,w1/w2
```

In Zeile 30 soll das Ergebnis von W1/W2 der Real-Variable »R« zugewiesen werden. »W1« und »W2« sind beide vom Typ Word, weshalb die Division in eben diesem Typ durchgeführt werden könnte. Doch das Ergebnis von W1 durch W2 wäre nicht 1.5, sondern 1, weil nicht mit Real-Zahlen gerechnet wurde. Dieses falsche Ergebnis würde dann »R« zugewiesen. Um dies zu verhindern, rechnet der Basic-Boss bei Zuweisungen grundsätzlich mindendestens mit dem Zieltyp, in diesem Fall Real. Darum stimmt auch der erste vom PRINT-Befehl in Zeile 30 ausgegebene Wert. Der zweite stimmt beim Compilat aufgrund der aufgezeigten Problematik nicht ganz, da es beim PRINT-Befehl keine Zielvariable gibt (darum kann der Mindesttyp für PRINT mit »£PRINTTYPE« eingestellt werden).

2.3.8 Der diffuse Wertebereich von Byte, Word und Integer

2.3.8.1 Byte und Word

Es wurde bisher behauptet, eine Byte-Variable könne nur Werte im Bereich von 0 bis 255 annehmen. Doch wie sieht es damit aus:

```
10 rem@ £byte a,b
20 a=5: b=-1
30 a=a+b
40 print a,b
```

Zum Wert der Variablen »A« (5) wird der Wert von »B« (–1) hinzuaddiert. Das Ergebnis müßte 4 sein. Und siehe, es funktioniert. »A« wird auch vom Compilat mit »4« angezeigt. Besonders interessant ist der Inhalt von »B«, denn –1 kann es nicht sein, weil dies nicht in den Wertebereich paßt. »B« wird in Zeile 40 mit »255« angezeigt. Doch warum 255? –1 ist um 1 kleiner als 0. Also muß man von 0 einen Schritt zurückgehen. Dazu ein anderes Beispiel: Wenn 10 um 1 vermindert wird, so erhält man 9. Nehmen wir einmal an, jemand sieht nur die hinterste Stelle und kennt darum nur die Zahlen von 0 bis 9. Wenn er die Ziffer 10 beobachtet, sieht er also nur die 0. Man sagt ihm, es werde jetzt 1 abgezogen und plötzlich wird aus der 0 eine 9. Als die Zahl am Ende des ihm bekannten Wertebereichs war (bei 0), sprang sie nach Abzug von 1 wieder auf den höchsten Wert des Wertebereichs (9). Genau so ist es beim Byte-Typ. Wenn man vom kleinsten Wert des Wertebereichs eine 1 abzieht, springt die Zahl auf den höchsten Wert. So erklärt sich die 255. Doch warum wird aus der 5 eine »4«. Ganz einfach: Zur 5 wurde 255 addiert (in Zeile 30) und das Ergebnis wäre 260. Da dies außerhalb des Wertebereichs von Byte liegt, kann man so oft 256 abziehen, bis man sich wieder im Wertebereich befindet. Und 260–256 ergibt »4«. Folglich kann man auch bei Byte in begrenztem Maße mit Vorzeichen rechnen. Ähnliches gilt für den Datentyp Word. Für ihn ist –1 identisch mit 65 535. Doch Vorsicht!

Nicht alle Operationen lassen dies zu. Es funktioniert bei Addition und Subtraktion und sogar bei der Multiplikation, nicht aber bei der Division. Es funktionieren »=« und »< >«, nicht aber »<», »<=«, »>«, »>=«. Im Typ Byte ist -1 definitionsgemäß größer als 100, denn -1 ist in diesem Typ identisch mit 255. Darum sollte man in diesen Fällen vorzugsweise mit »=« und »< >« arbeiten.

2.3.8.2 Der Integer-Typ und seine Ähnlichkeit zu Word

Keine solchen Probleme hat man beim Integer-Typ. Er kann Werte zwischen -32768 und +32767 annehmen. In diesem Bereich arbeiten alle Operationen einwandfrei und erwartungsgemäß. Dies gilt auch für »<», »<=«, »>«, »>=« (diese Operationen sind im Integer-Typ aber relativ langsam) und die Division. Da dieser Typ ebenfalls wie Word genau zwei Byte an Speicher belegt, liegt die Vermutung nahe, daß sie sich irgendwie ähneln. Und in der Tat sind sie sich nicht nur ähnlich, sondern eigentlich identisch. Denn ob man einer Integer-Variablen den Wert -1 zuweist, oder einer Word-Variablen ist egal. In beiden Fällen steht im Speicher exakt dasselbe. Wodurch unterscheiden sich diese Typen dann überhaupt? Nur durch die Operationen »<», »<=«, »>«, »>=«, die Division und die Umwandlung in einen String (bei PRINT oder STR$) oder nach Real. Ansonsten sind diese Typen völlig gleich! Damit ist zum Beispiel auch »I=55296: POKE I,1« mit »I« als Integer-Variable möglich. Da der Datentyp Word der wichtigere ist, wird bei einem Typkonflikt übrigens willkürlich nach Word gewandelt. Eine solche Umwandlung von Integer nach Word oder von Word nach Integer verbraucht aber weder Zeit noch Speicher, da letztendlich keinerlei Umwandlung stattfindet.

2.3.9 Typen bei Operationen, Funktionen und Befehlen

Die von Operationen, Funktionen und Befehlen erwarteten und gelieferten Typen sollte man kennen, schon allein deshalb, damit keine unnötigen Umwandlungen vorgenommen werden, die Zeit und Speicher verschwenden.

2.3.9.1 Typen bei Befehlen

Viele Befehle des Basic V2 erwarten einen oder mehrere Parameter. Diese Parameter sind beim Interpreter immer vom Typ Real. Der Compiler aber erwartet einen bestimmten zweckmäßigen Typ. Wenn der Benutzer einen Ausdruck eines anderen Typs angibt, so wird eine Umwandlung vorgenommen. Wenn ein bestimmter Typ erwartet wird, dann ist dieser Typ hier einfach an die Stelle des Parameters gesetzt.

2.3.9.2 Speicherbefehle

POKE	Word, Byte
SYS	Word

2.3.9.3 Befehle zur Programmsteuerung

GOTO	Number
GOSUB	Number
ON Byte GOTO	Number
ON Byte GOSUB	Number

»Number« ist eine Zahl vom Typ Word und keine Variable.

FOR N=A TO B STEP C
»N« muß eine Variable des Typs Byte, Word, Integer oder Real sein (also numerisch). A,B, und C sollten vom selben Typ sein.

NEXT N
»N« muß dieselbe Variable wie bei FOR sein.

IF A THEN ...	Number
IF A GOTO	Number
IF A THEN	Number

»A« kann ein Ausdruck von jedem beliebigen Typ sein.

2.3.9.4 Ein- und Ausgabe (im weiteren Sinne)

PRINT A
INPUT V
GET V und READ V

OPEN	Byte, Byte, Byte, String
CLOSE	Byte
PRINT #	Byte, A
GET #	Byte, V
INPUT #	Byte, V
LOAD	String, Byte, Byte
SAVE	String, Byte, Byte
CMD	Byte

»A« kann ein Ausdruck von jedem Typ sein.
»V« ist eine Variable von beliebigen Typ.

2.3.9.5 Weitere Befehle

DIM V	(Number,...)
DATA	D, D, D
RESTORE	Word
WAIT	Word, Byte, Byte

Die Daten »D« sind normale Zeichenketten oder Zahlen

2.3.9.6 Typen bei Funktionen

Funktionen unterscheiden sich von Befehlen dadurch, daß sie Ergebnisse liefern. Auch diese Ergebnisse sind von einem bestimmten Typ. SIN, COS, TAN, ATN, LOG, EXP, RND, SQR, USR erwarten jeweils eine Real-Zahl und liefern auch eine solche (zum Beispiel »Real=SIN(Real)«. SGN, INT, ABS lassen sich auf alle numerischen Typen anwenden. Das Ergebnis ist jeweils eine Zahl in diesem Typ.

Byte = PEEK (Word)
Word = FRE (X)
Byte = POS (X)

»X« ist völlig unbedeutend (Dummy-Parameter)

Byte = ASC (String)
Real = VAL (String)
String = CHR$ (Byte)
String = STR$ (A)

»A« ist ein beliebiger numerischer Ausdruck

Byte = LEN (String)
String = LEFT$ (String, Byte)
String = RIGHT$ (String, Byte)
String = MID$ (String, Byte, Byte)

In diese Sparte passen auch Arrays. Sie erwarten Parameter vom Typ Word und liefern den Typ der Array-Variable. Bei eindimensionalen Arrays vom Typ Byte, Boolean, Word oder Integer wird auch der Parametertyp Byte erwartet, falls die Dimension des Arrays entsprechend klein ist (dann kann eine effizientere Adressierungsart angewandt werden).

2.3.9.7 Typen bei Operationen

Die folgenden Operatoren liefern denselben Typ, mit dem sie versorgt werden: Plus (+), Minus (–), AND und OR verarbeiten sämtliche numerischen Typen. Mal (*) und Durch (/) verarbeiten Word, Integer und Real. Potenzieren kann man nur mit Real-Zahlen. Die Negation (–) und NOT verarbeiten alle Typen von Byte bis Real. AND, OR und NOT verarbeiten ebenfalls alle numerischen Typen und zusätzlich noch Boolean. Es gibt auch eine Gruppe von Operatoren, die nicht den Typ liefern, den sie erwarten. Dies sind die Vergleichsoperatoren. Sie verarbeiten alle Typen (auch String) und liefern grundsätzlich den Typ Boolean.

2.3.9.8 Typen der Systemvariablen

Der C64 kennt die Systemvariablen TI, TI$ uns ST. TI ist vom Typ Real, TI$ vom Typ String und ST wird als Byte behandelt.

2.3.10 Der Datentyp »Constant«

Pascal und Modula bieten dem Programmierer die Möglichkeit, schwer interpretierbare Zahlen durch sinnvolle Namen zu ersetzen. Diesen Vorzug besitzt Basic nicht. Hier muß man stattdessen Variablen verwenden, wenn man keine Zahlen schreiben will. Das ist besonders bedauerlich, weil die Effizienz der Compilate bei Verwendung von konstanten Zahlen statt Variablen deutlich zunimmt. Der Code wird schneller und kürzer. »POKE COL,1« ist kompiliert zum Beispiel drei mal so lang und langsam wie »POKE 53281,1«. Darum habe ich den Basic-Boss um den Datentyp Constant bereichert. Eine Konstante wird wie jede andere Variable deklariert, wobei als Datentyp »CONSTANT« anzugeben ist. Es können Variablen der Form »A«, »A%« und »A$« als Konstanten deklariert werden. Wie jeder anderen Variablen kann einer Konstanten im Programm ein Wert zugewiesen werden, indem man hinter den Variablennamen ein Gleichheitszeichen setzt, dem der gewünschte Wert folgt. Auf diese Weise wird die Kompatibilität zum Interpreter gewahrt. Allerdings darf einer Konstanten nur ein einziges Mal ein Wert zugewiesen werden. Dies muß zudem vor ihrer Benutzung geschehen (also in einer früheren Programmzeile). Darum sollte man den Konstanten am besten am Programmanfang ihren Wert zuweisen. Dann ist die Konstante einsatzbereit. Sie kann im nachfolgenden Programm beliebig oft verwendet werden und wird dann jeweils durch den ihr zugewiesenen Wert ersetzt.

Ein Beispiel:

```
10 rem constant col
20 col=53281
30 poke col,1
```

Dieses Programm ist aus der Sicht des Compilers exakt identisch mit:

```
30 poke 53281,1
```

Im ersten Programm ist die Zeile 30 aber wesentlich besser lesbar. Sie sollten sich bei der Verwendung von Konstanten darüber im klaren sein, daß diese von Haus aus keinen bestimmten Typ besitzen. Bei der Zuweisung eines Wertes an eine Konstante wird eigentlich nur der Text gelesen und gemerkt. Darum kann man einer Konstante alles mögliche zuweisen (zum Beispiel »C="HALLO"« oder »C=4+1.23*5«). Wenn diese Konstante später im Programmtext auftaucht, dann wird der ihr zugewiesene Text geklammert und eingesetzt. Erst dann wird der Ausdruck ausgewertet. Wenn sich im Konstantentext Fehler befanden, so werden diese folglich erst jetzt gemeldet. Die Klammerung ist übrigens notwendig. Man nehme an, »C« sei eine Konstante und sei so bestimmt worden: »C=1+2«. Wenn später der Ausdruck »A=C*3« auftaucht, dann würde sich ohne Klammerung »A=1+2*3« ergeben und das Ergebnis wäre 7. Gemeint war aber »A=(1+2)*3«, was 9 ergibt. Konstanten können geschachtelt werden. Wenn zum Beispiel VIC und SPR Konstanten sind, dann lassen sich diese auch so belegen: »VIC=53248: SPR=VIC+21«. Wenn SPR im folgenden Programmtext verwendet wird, dann wird zunächst SPR ersetzt, dann VIC und es ergibt sich für »POKE SPR,255« der Ersetzungstext »POKE ((53248)+21),255«.

2.4 Beschreibung der Direktiven

Die Einstellungen und Eigenschaften des Compilers werden ausschließlich über Compilerdirektiven verändert, die im Basic-Text des Quellprogramms stehen. Dies erhöht die Flexibilität und den Bedienungskomfort des Basic-Boss, da man programmspezifische Parameter nicht wie bei manchen anderen Compilern bei jedem Compilerlauf neu einstellen muß. Bei der Anwendung von Compilerdirektiven sollten Sie sich immer darüber im klaren sein, daß der Compiler den Basic-Text Zeile für Zeile von vorne nach hinten durcharbeitet und sich nicht um die tatsächliche Abarbeitungsreihenfolge unter dem Interpreter kümmert (die von GOTO und GOSUB bestimmt wird). Die Direktiven sollten darum grundsätzlich am Anfang eines Programmes stehen. Eine Reihe von Direktiven kann im Gegensatz dazu beliebig oft innerhalb des Programms verwendet werden. Wenn dies der Fall ist, so wird es in der Beschreibung der Direktive erwähnt. Im folgenden werden die Compilerdirektiven nach ihrer Funktion geordnet erklärt. Sie sind in Großbuchstaben geschrieben und es muß ihnen jeweils noch ein Pfund (£) vorangestellt werden, damit sie lauffähig sind. Wenn Sie nicht unbedingt alle Möglichkeiten

des Basic-Boss bis ins Letzte ausreizen wollen, brauchen Sie sich nur die Befehle anzusehen, die als wichtig bezeichnet werden.

2.4.1 Steuerung der Datentypen

Folgende Befehle sind sehr wichtig:

(£BOOLEAN), £BYTE, £WORD, £INTEGER, £REAL, £STRING

Diesen Befehlen folgt eine Variablenliste der Form »A,B%,C,D$« mit beliebig vielen Elementen. Statt der Variablen können auch »%«, »$« oder »#« für alle Variablen der Form »A%«, »A$« oder »A« stehen. Die betroffenen Variablen werden zu diesem Typ deklariert. Hinter einer Variable kann ein Gleichheitszeichen, dem wiederum »FAST« oder eine Adresse folgen muß. Diese Befehle müssen am Anfang des Programms stehen. Mehr hierzu finden Sie unter Kapitel 2.3 »Datentypen«. Den nächsten Befehl sollte man ebenfalls kennen, wenn man viel mit DATA arbeitet:

£DATATYPE Typ

So wird der Typ der Daten des DATA-Befehls festgelegt. Standardmäßig sind alle Daten vom Typ String und werden auch in diesem Typ im Compilat abgelegt. Denn der Typ String kann vom READ-Befehl in alle anderen Typen umgewandelt werden. Wenn die Daten aber in Wirklichkeit zum Beispiel Zahlen zwischen 0 und 255 sind (was sehr häufig der Fall ist) und einer Variablen vom Typ Byte zugedacht sind, so wäre es sicher sinnvoller, die Daten bereits in diesem Typ abzulegen. Dies geschieht, nachdem die Direktive »£DATATYPE Byte« gefunden wurde. Danach sollten die Daten hinter dem DATA-Befehl Zahlen zwischen »0« und »255« sein. Sie werden dann direkt im Typ Byte abgelegt, was sehr viel Speicher spart (zirka 70% bis 80%) und die Ablaufgeschwindigkeit des Compilats drastisch erhöht. Man kann statt »Byte« auch alle anderen Datentypen einsetzen, wobei diese dann entsprechend mehr Speicherplatz benötigen (»REAL« zum Beispiel benötigt fünf Byte pro Datenwert und spart damit nicht immer Speicher gegenüber dem Stringtyp, dafür aber Zeit). Diese Direktive kann beliebig oft im Programm angewandt werden, so wie es die unterschiedlichen Daten erfordern. Dabei ist jedoch

zu beachten, daß der READ-Befehl die numerischen Typen nicht ineinander umwandeln kann. Darum steigt das Compilat mit einem »type mismatch error« aus, wenn Sie zum Beispiel Byte-Daten in eine Real-Variable einlesen wollen. Das Problem kann gelöst werden, indem man die Byte-Daten zunächst in eine Byte-Variable einliest und diese Byte-Variable anschließend einer Real-Variablen zuweist, was immer noch schneller und vor allem platzsparender ist als die Benutzung normaler Stringdaten.

Ein Beispiel:

```
10 rem@ £byte a,b,c
20 read a,b,c
30 read x$,y$,z$
40 rem@ £datatype byte
50 data 10,200,50
60 rem@ £datatype string
70 data dies,sind,strings
```

Die folgenden Direktiven sind ziemlich unwichtig und werden selten benötigt.

£PRINTTYPE Typ

Es wird der Mindesttyp der Berechnung im PRINT-Befehl auf den angegebenen Typ festgelegt. Nach »£PRINTTYPE Real« werden also alle Operationen im PRINT-Befehl mindestens im Real-Typ durchgeführt. Wenn »W1« und »W2« vom Typ Word sind und die Werte »5« und »2« enthalten, so liefert der Befehl »PRINT W1/W2« nicht mehr das falsche Ergebnis »2«, sondern »2.5«. Standardmäßig ist der Typ Boolean.

£BOOLEANTYPE Typ

Dieser Befehl legt den Mindesttyp bei der Berechnung von IF-Ausdrücken fest (zum Beispiel »IF A+3.5=B THEN ...«). Dieser Mindesttyp ist standardmäßig Boolean.

£CALCTYPE Typ

Hiermit wird der Mindesttyp bei Berechnungen allgemein festgelegt. Normalerweise ist er auf Boolean gesetzt. Um die volle Kompatibilität zu Basic herzustellen, kann man ihn auf Real setzen (mit »£CALCTYPE real«), was die Programme aber unnötig verlängert und stark verlangsamt.

2.4.2 Optimierungsmöglichkeiten

Die nächsten zwei Befehle sollten Sie kennen, wenn Sie schnelle Programme schreiben wollen:

£FASTFOR und £SLOWFOR

Die FOR-NEXT-Schleife arbeitet im Compilat sehr umständlich und langsam, da sie genau so funktionieren soll, wie in Basic. Deshalb bietet der Basic-Boss eine alternative FOR-NEXT-Schleife an, die wesentlich schneller ist, dafür aber ein paar Einschränkungen aufweist. Dies sind folgende:
FOR und NEXT müssen im Programmtext in richtiger Reihenfolge stehen und auf jedes FOR muß genau ein NEXT folgen (ähnlich wie in Pascal, Modula oder C). Folgende Konstruktionen lassen den Basic-Boss darum schmollen, wenn er schnelle Schleifen erzeugen soll:

```
10 goto 30
20 next i: end
30 for i=1 to 100
40 print i: goto 20
```

oder so etwas:

```
10 for r=1 to 100
20 print i;
30 if i<50 then next i
40 print "abc"
50 next i
```

Eine weitere Einschränkung besteht beim Typ der Zählvariable. Sie darf nur vom Typ Byte oder Word sein. Andernfalls benutzt der Basic-Boss die langsame Schleife. Auch beim STEP-Wert ist nicht alles erlaubt: Wenn er angegeben wird, so muß er direkt angegeben werden, das heißt als Zahl und nicht als Variable. Er darf allerdings sowohl positiv als auch negativ sein. Außerdem darf sich der Wert der Endvariable nicht verändern. Bei »FOR I=A TO E« muß der Wert in »E« während der Ausführung der Schleife immer gleich bleiben, wenn die Schleife im Compilat dasselbe machen soll wie unter dem Interpreter (es sei denn, »E« ist ein Ausdruck). Trotz all dieser Einschränkungen läßt sich der FASTFOR-Befehl auf die allermeisten Schleifen anwenden. Normalerweise produziert der Basic-Boss langsame Schleifen (der Kompatibilität wegen). Wenn er aber die Direktive »£FASTFOR« findet, erzeugt er von da ab schnelle Schleifen. Wenn er dann wieder auf »£SLOWFOR« stößt, erzeugt er wieder langsame, aber vollkompatible Schleifen. Wenn man nur eine einzige Schleife schneller machen will, dann kann man also vor diese Schleife »£FASTFOR« und dahinter »£SLOWFOR« schreiben. Man sollte beachten, daß bei »£SLOWFOR« alle Schleifen mit NEXT abgeschlossen sind, da der Basic-Boss ansonsten einen Fehler meldet. Man sollte übrigens grundsätzlich die Schleifenvariable hinter NEXT angeben, da der Compiler dann Fehler besser aufdecken kann. Schachtelungen sind natürlich auch bei schnellen Schleifen erlaubt. Es ist bei FASTFOR-Schleifen im Gegensatz zu langsamen Schleifen außerdem völlig unproblematisch, aus der Schleife mit GOTO herauszuspringen. Die nächsten vier Befehle sind für die Optimierer unter Ihnen gedacht und nicht von essentieller Bedeutung.

£LONGIF und £SHORTIF

Es gibt für den Compiler zwei Möglichkeiten, eine IF-Verzweigung zu konstruieren. Das eine Verfahren erlaubt beliebig viele Befehle hinter dem THEN, das andere nicht. Dafür spart man mit dem zweiten Verfahren aber bei jedem IF-Befehl drei Byte Speicher und der Code wird etwas schneller. Standardmäßig wendet der Compiler das LONGIF-Verfahren an, das eine beliebig große IF-Verzweigung erlaubt. Mit »£SHORTIF« schaltet man auf die Sparversion um, die in den allermeisten Fällen aber ausreicht. Darum würde ich empfehlen, grundsätzlich mit »£SHORTIF« zu programmieren. Wenn der Basic-Boss dann doch einmal bei einer IF-Zeile streikt (er meldet einen Fehler), dann kann man vor dieser IF-Zeile mit »£LONGIF« auf die

längere Version umschalten und dahinter mit »£SHORTIF« wieder auf die kürzere. Die folgenden zwei Befehle können beim Austesten wichtig sein:

£SLOWARRAY und £FASTARRAY

Wenn ein Feld (Array) zum Beispiel mit »DIM A(20)« dimensioniert ist, so erlaubt es der Compiler bei einfachen und schnellen Arrays, daß das Basic-Programm auch auf nicht existente Array-Elemente zugreift. Die Anweisung »X=30: A(X)=1000« wird also anstandslos abgearbeitet, wodurch andere Variablen oder Speicherinhalte zerstört werden. Zum Ausgleich dafür sind solche Arrays aber extrem schnell. Wenn man dennoch auf Nummer Sicher gehen will, so kann man dies mit »£SLOWARRAY« tun, was das Programm dann stark verlangsamt. »£FASTARRAY« schaltet die schnelle Array-Verwaltung wieder ein. Beide Befehle können mehrmals an beliebiger Stelle im Programm stehen, weshalb man die Sicherheit gezielt dosieren kann.

2.4.3 Festlegung der Startadressen

Diese Befehle sollte man nur dann anwenden, wenn man gewisse Grundkenntnisse bezüglich der Speicheraufteilung des C64 besitzt. Ansonsten kann man sie ignorieren. Für gewöhnlich wird das Compilat in ein einziges File gespeichert, dessen Name der Basic-Boss aus dem Namen des Quellprogramms erhält, indem er ihm ein »+« voranstellt. Das Compilat wird an den normalen Basic-Start (2049) inclusive SYS-Zeile kompiliert, so daß man es mit »RUN« starten kann. Es geht aber auch anders. Man kann das Compilat an einen beliebigen Ort im Speicher legen. Man kann es sogar in seine Bestandteile aufspalten und diese an beliebige Speicheradressen legen und unter beliebigem Namen auf Diskette schreiben. So ist es sogar möglich, mehrere Compilate gleichzeitig im Speicher zu halten, die sich gegenseitig aufrufen, wenn man sie an unterschiedliche Adressen legt. Zu all dem kann noch ein gewöhnliches Basic-Programm kommen, das die Compilate möglicherweise als Unterroutinen benutzt und steuert.
Die Speicheraufteilung des Compilats sieht normalerweise so aus: Es besteht aus drei Teilen. Dies sind der Programmteil, der Routinenteil und der Datenteil. Der Programmteil stellt das eigentliche Maschinenprogramm dar, das aus dem Basic-Programm erzeugt wurde. Dieser Programmteil benötigt für seine Arbeit Unterroutinen, die der Basic-Boss einzeln aus seiner

Bibliothek holt und hinter den Programmteil schreibt, wobei der Compiler sie an den Adreßbereich anpaßt (insgesamt heißt dieser Vorgang »Linken«). Dann folgt ein Datenbereich, in dem Informationen über Array-Variablen und eine Tabelle für den READ-Befehl stehen. Schließlich folgen der Variablenspeicher, der Bereich für Hilfespeicher und letztendlich der Stringspeicher. Diese Reihenfolge scheint zwar die vernünftigste zu sein, entspricht aber nicht der Realität. Oder genauer: Sie stimmt nur in Pass 1. In Pass 2 erzeugt der Compiler zunächst Routinen- und Datenbereich und erst danach den Programmbereich. Dies erfordert umfangreiche Umrechnungen, hat dafür aber einen wesentlichen Vorteil: Der Routinenbereich steht im Compilat ganz vorne. Damit ist im Normalfall sichergestellt, daß die Routinen im unteren 40-Kbyte-Bereich liegen. Und genau dort müssen sie sein, denn sie müssen unbedingt in einem Speicherbereich stehen, der nicht vom ROM oder von anderem überlagert wird.
Die Lage der Bereiche läßt sich mit folgenden Befehlen ändern. Dabei ist zu beachten, daß sich die einzelnen Bereiche nicht überschneiden. Wenn die Lage eines Code-Bereichs geändert wird, dann wird er in eine eigene Datei geschrieben. Eine alte Datei gleichen Namens wird standardmäßig gelöscht (mit dem SCRATCH-Befehl).

£ROUTSTART Adresse

legt den Anfang für den Routinenbereich fest. Die Routinen dürfen nicht unter das ROM oder unter den I/O-Bereich gelegt werden. Der Dateiname dieses Teils ergibt sich standardmäßig aus »R+« und dem Namen der Quelldatei. Die SYS-Zeile ist Teil dieses Bereichs. Darum schaltet der Befehl außerdem die SYS-Zeile ab, falls sie nicht explizit mit »£SYSON« aktiviert wurde. Dann kann der Routinenbereich direkt mit einem SYS-Befehl angesprungen werden, wenn das Compilat gestartet werden soll. Standardmäßig liegt der Routinenbereich bei 2049, also am Beginn des Basic-Speichers.

£DATASTART Adresse

legt die Startadresse für den Datenbereich fest. Der Dateiname des Datenbereichs ergibt sich aus »D+« und dem Namen der Quelldatei.

£PROGSTART Adresse

legt die Startadresse des eigentlichen Programms fest. Es kann überall hingelegt werden, zum Beispiel auch nach $D000, sofern das Compilat im »£ALLRAM«-Modus läuft. Der Dateiname ergibt sich aus »P+« und dem Namen der Quelldatei.

£VARSTART Adresse

bestimmt die Adresse des Variablenspeichers.

£HELPSTART Adresse

bestimmt den Anfang des Hilfespeichers.

£HEAPSTART Adresse

und

£HEAPEND Adresse

bestimmen die Lage des Stringspeichers. Die Adresse bei »£HEAPEND« gibt die erste nicht mehr benutzte Speicherstelle an. Wenn bei »£HEAPEND« »0« angegeben wird, so wird überhaupt kein Stringspeicher reserviert, was sinnvoll ist, wenn Sie keine Stringoperationen ausführen wollen. Variablen-, Hilfs- und Stringspeicher können wie der Programmbereich überall hingelegt werden. Normalerweise werden alle Bereiche direkt aneinandergehängt. Falls Sie sich eine eigene Basic-Boss-Version mittels »£SETPREFERENCES« erzeugt haben, die die Bereiche nicht aneinanderhängt, so können Sie dies mit der Angabe von »0« als Startadresse eines Bereichs erzwingen.

2.4.4 Die Dateien

Die Filenamen der Ausgabedateien können Sie beliebig festlegen, falls Ihnen die Standardnamen nicht zusagen.

£ROUTFILE Gerät, "Dateiname"

So wird der Name der Datei für den Routinenteil festgelegt, in die für gewöhnlich auch der Datenteil und der Programmteil geschrieben werden. Das Gerät ist sinnvollerweise entweder »8« oder »9«, normalerweise wird man es weglassen (»£ROUTFILE "Dateiname"«). Hinter dem Gerät könnte man auch noch die Sekundäradresse angeben, was aber in den wenigsten Fällen sinnvoll sein dürfte (nur bei besonders umständlichen RAM-Disks oder ähnlichem).

£PROGFILE Gerät, "Dateiname"

und

£DATAFILE Gerät, "Dateiname"

funktionieren genauso für den Programmbereich und den Datenbereich. Dem Namen einer Datei kann unbedenklich ein Klammeraffe (@) gefolgt von einem Doppelpunkt vorangestellt werden. Der Basic-Boss erkennt dies und löscht dann die alte Datei vor dem Öffnen der neuen mit dem SCRATCH-Befehl der Floppy. Dies ist notwendig, weil das Überschreiben mit dem Klammeraffen aufgrund eines Fehlers in der Floppy nicht funktioniert und normalerweise grundsätzlich vermieden werden sollte, da sonst Dateien zerstört werden können. Wenn man dem Dateinamen keinen Klammeraffen voranstellt, dann wird eine alte Datei gleichen Namens nicht gelöscht und es tritt eventuell ein Floppy-Fehler auf. Es gibt noch einen weiteren Dateibefehl:

£SOURCEFILE Gerät, Sekundäradresse, "Dateiname"

Dieser Befehl scheint reichlich unsinnig. Er legt Name, Gerät und Sekundäradresse der Quelldatei fest, nachdem dieselbige bereits geöffnet ist (er wird ja aus ihr gelesen). Darum ist er nur vor einem »£SETPREFERENCES«-Befehl sinnvoll. Sekundäradresse und/oder Gerät können wie üblich weggelassen werden.

2.4.5 Protokollierung der Compiler-Tätigkeit

Auch die folgende Direktive ist nicht von überragender Bedeutung, sollte aber trotzdem in keinem Compiler oder Assembler fehlen.

£PROTOCOL Gerät, Sekundäradresse, "Dateiname"

Die Fehlermeldungen und sonstige Daten (zum Beispiel die Speicheradressen) erscheinen normalerweise nur auf dem Bildschirm. Wenn man sie auf den Drucker leiten will, so genügt die Direktive »£PROTOCOL«. Wenn der Drucker eine andere Geräteadresse als »4« besitzt (zum Beispiel Plotter 1520), dann sollte man diese mit »£PROTOCOL Gerät« angeben. An das Gerät kann man noch die Sekundäradresse anhängen und eventuell auch einen Dateinamen (Dieser steuert zum Beispiel beim Görlitz-Interface den Zeilenabstand). »£PROTOCOL 8, "FEHLER",S,W« sendet die Fehlermeldungen dagegen in die Datei »FEHLER« auf Diskette (auch hier kann man unbedenklich den Klammeraffen verwenden). Diese Datei ist eine simple ASCII-Datei und kann zum Beispiel mit dem Programm »ERRORREAD« (auf der Programmdiskette) wieder gelesen werden. Wenn »£PROTOCOL« verwendet wurde, dann hält der Compiler beim Auftreten von Fehlern nicht mehr an. Mit »£PROTOCOL 0« erzeugen Sie diesen Effekt, ohne daß irgendeine Ausgabe erfolgt.

2.4.6 Speicherverwaltung

Der Basic-Boss erlaubt es, den gesamten Speicher des C64 zu nutzen. »PEEK« und »POKE« werden dabei etwas langsamer. Wenn Sie das nicht stört, dann setzen Sie einfach die Direktive »£ALLRAM« an den Programmanfang und die Sache ist erledigt. Falls Sie aber optimale Geschwindigkeit wünschen, dann sollten Sie sich mit den Befehlen »£RAM« und »£ROM« beschäftigen.

£ALLRAM

muß am Anfang des Programms stehen und weist den Basic-Boss an, den gesamten Speicher für das Programm zu nutzen. Zudem wird das Stringspeicherende automatisch an das Speicherende gesetzt, falls kein »£HEAPEND«-Befehl im Programm steht. Im ALLRAM-Modus sind grundsätzlich die gesamten 64 Kbyte RAM des C64 eingeschaltet (normalerweise sind nur die untersten 40 Kbyte eingeschaltet und ein kleiner Bereich ab Adresse 49152). Allerdings muß dann bei allen PEEK-, POKE- und SYS-Befehlen zuerst auf das ROM umgeschaltet werden, damit sie in jedem Fall dasselbe Ergebnis erbringen wie beim Interpreter. Dies verlangsamt das Programm nicht unbeträchtlich. Wenn man mit diesen Befehlen aber sowieso nur den RAM-Speicher ansprechen will und nicht das ROM, den I/O-Bereich oder den Farbspeicher, dann kann man die Umschaltung mit £RAM unterbinden. Jeder POKE-, PEEK-, oder SYS-Befehl bezieht sich dann auf den RAM-Speicher. Wenn wieder auf das ROM oder die I/O-Bausteine zugegriffen werden soll, muß man die Umschaltung zuvor mit £ROM wieder anschalten. Wenn man diesen Befehl vergißt, dann wird zum Beispiel nicht die Bildschirmfarbe gesetzt, sondern der Stringspeicher gestört und die Garbage-collection läuft Amok. Diese beiden Befehle sind nur bei »£ALLRAM« sinnvoll. »£RAM« und »£ROM« können beliebig oft im Programm verwendet werden. Ein Beispielprogramm (»RAMROM«) befindet sich auf der Programmdiskette. Auch im Programm »RAMLOAD« wird von »£ALLRAM« und »£RAM« Gebrauch gemacht. Eine effizientere Möglichkeit des Zugriffs auf das ROM im ALLRAM-Modus bieten die Befehle »←RAM«, »←ROM« und »←IOROM«, die aber nur der Profi einsetzen sollte (siehe unter 2.5 »Neue Befehle und Funktionen«).

£BASICRAM

ist das Gegenstück zu »£ALLRAM« und bewirkt, daß nur der normale Basic-Speicher für Programm und Variablen benutzt wird. Wenn das Stringende (»£HEAPEND«) noch nicht gesetzt wurde, wird es auf 40960 herabgesetzt (Ende des Basic-Speichers). Dies ist der Normalzustand.
Im ALLRAM-Modus kann das Compilat über den Basic-Bereich hinausreichen und länger als 154 Blocks sein. Dann belegt es auch das RAM unter dem ROM. Eine größere Länge als 201 bis 202 Blocks ist nur deshalb nicht möglich, weil die Lade-Routine des C64 dann den I/O-Bereich überschreibt, was sich optisch an einer drastischen Änderung der Bildschirmanzeige erkennen läßt. In diesen Bereich kann ein Programm nur mit dem Compilat von »RAMLOAD« geladen werden. Dabei darf RAMLOAD selbst aber nicht überschrieben werden.

2.4.7 Die SYS-Zeile

£SYSON und £SYSOFF

Die SYS-Zeile ermöglicht es, das Compilat wie ein normales Basic-Programm zu laden und zu starten. Sie kann mit »£SYSOFF« unterbunden werden. Dies ist jedoch meist nicht notwendig, da der »£ROUTSTART«-Befehl die SYS-Zeile selbsttätig abschaltet, so daß das Programm mit »SYS Startadresse« vom Direktmodus aus oder von einem anderen Programm gestartet werden kann. Mit »£SYSON« kann man die SYS-Zeile erzwingen. Folgende Befehle haben nur rein kosmetische Bedeutung:

£SYSNUMBER Nummer

bestimmt die Zeilennummer der SYS-Zeile.

£SYSTEXT "Text"

legt den Text in der SYS-Zeile fest.

2.4.8 Sonstige Direktiven

£LINEON und £LINEOFF

schalten die Zeilenaktualisierung an und aus. Normalerweise ist sie abgeschaltet, da sie Speicherplatz verbraucht und das Programm verlangsamt. Wenn sie angeschaltet ist, werden die während des Programmlaufs auftretenden Fehler mit der richtigen Zeilennummer angezeigt, sonst nicht. Die nun folgenden Befehle sind für den gelegentlichen Benutzer von geringerer Bedeutung:

£LONGNAME und £SHORTNAME

In Basic sind bei Variablen beliebig lange Namen erlaubt. Trotzdem unterscheidet der Interpreter nur die ersten zwei Stellen. »KUCKUCK« und »KUHFLADEN« sind für den Interpreter also genau dasselbe. Der Compiler ist aber auf beliebig lange Variablen ausgelegt. Aus Kompatibilitätsgründen unterscheidet er trotzdem nur zwei Stellen. Mit »£LONGNAME« kann man diesen Mangel beheben, wenn man in Kauf nimmt, daß das Programm unter dem Interpreter dann nicht mehr einwandfrei läuft. »£SHORTNAME« schaltet wieder auf zweistellige Namen. Beim Benutzen langer Namen sollte man darauf achten, daß kein Basic-Befehl in diesen Namen enthalten ist, da dies zwangsläufig zu einem Fehler führt.

£IGNORE und £USE

»£IGNORE« weist den Compiler an, alles nachfolgende zu ignorieren. Erst wenn er auf »£USE« trifft, untersucht er das Basic-Programm wieder. Diese Funktion ist sinnvoll, wenn man einige Teile des Basic-Programms vor dem Compiler verstecken will. Das kann sehr nützlich sein, wenn die Lauffähigkeit unter dem Interpreter erhalten bleiben soll und man trotzdem spezielle Compilerfunktionen ausnutzen will.

Ein Beispiel:

```
10 print "fuer compiler und interpreter"
20 rem@ £ignore
30 print "nur fuer interpreter"
```

```
40 goto 70
50 rem@ £use
60 print "nur fuer compiler"
70 print "wieder fuer beide"
```

£SETPREFERENCES

Wenn man seine Lieblingseinstellungen nicht in jedem Programm per Compilerdirektive neu setzen will (zum Beispiel Programmstart, Standarddatentyp und so weiter), dann kann man ein kleines Programm schreiben, das die erwünschten Einstellungen setzt und anschließend den »£SETPREFERENCES«-Befehl einsetzt. Die momentanen Einstellungen des Basic-Boss werden dann in die Standardeinstellungen kopiert und der Compiliervorgang wird abgebrochen. Wenn man den Basic-Boss jetzt mit < ← > neu startet, kann man einen Stern (»*«) gefolgt von einem Namen eingeben, unter dem Ihr individueller Basic-Boss anschließend gespeichert wird (nicht auf die Originaldiskette!). In Zukunft können Sie dann diese Basic-Boss-Version statt der Standardversion benutzen und müssen die am häufigsten gebrauchten Einstellungen nicht jedesmal neu setzen. Ein Beispielprogramm hierzu finden Sie unter dem Namen »PREFERENCES« auf der Diskette.

£CODE Wert, Wert, ...

Dieser Befehl ist nur für Kenner der Maschinensprache gedacht. Dem Befehlswort folgen beliebig viele mittels Komma getrennte Werte. Dies können Byte-Werte sein. Es können auch Word-Werte sein, wenn man dem Zahlenwert ein »w« vorausstellt. Auch Zeichenketten sind möglich, wenn man diese in » " « einschließt. Diese Bytes oder Words oder Zeichenketten werden an der aktuellen Position in das Compilat eingefügt. Auf diese Weise kann man sämtliche Maschinensprachebefehle des 6510 erzeugen. Man braucht dabei nicht auf den Erhalt der Register-Werte (A,X,Y) zu achten, da der Basic-Boss dies nicht verlangt. Wenn man den »£CODE«-Befehl unvorsichtig einsetzt, dann ist das Compilat anschließend aber nicht lauffähig. Ein paar Beispiele: »£CODE 234,234« fügt zwei NOP-Befehle ein, was eine kleine Zeitverzögerung bewirkt. »£CODE 0« setzt einen Breakpoint. »£CODE 96« ist identisch mit

»RETURN«. »£CODE 108,w40962« führt »JMP ($A002)« aus und »£CODE 2« erzeugt einen sicheren Systemabsturz.

£INITOFF und £INITON

Sämtliche Variablen werden bekanntlich zu Beginn eines Programms auf »0« gesetzt (FAST- und Adreßvariablen ausgenommen). Dies kann man mit »£INITOFF« verhindern. Dieser Befehl muß unbedingt am Anfang eines Programms stehen. Wenn er vor Beginn der Code-Erzeugung gefunden wird, dann werden keine Variablen initialisiert und haben beim Start des Compilats einen zufälligen Wert. Wenn dieser Befehl verwendet wird, sollte man auf Strings verzichten, da diese eine Initialisierung unbedingt benötigen. »£INITON« schaltet die Initialisierung wieder ein.

2.5 Neue Befehle und Funktionen

Der Basic-Boss versteht einige neue Befehle und Funktionen. Die Befehle werden mit einem »←« eingeleitet und die Funktionen mit dem Klammeraffen (@). Da der Basic-Interpreter beides nicht verarbeitet, kann man »£IGNORE« und »£USE« hierbei nutzbringend einsetzen und für Interpreter und Compiler stellenweise verschiedene Befehlsfolgen vorsehen, damit das Programm nach wie vor einwandfrei unter dem Interpreter ausgetestet werden kann.

2.5.1 Ein-/Ausgabe

←LOAD "DATEINAME", GERÄT, ADRESSE

Dieser Befehl lädt vom angegebenen Gerät eine Datei an die angegebene Adresse. Falls die Adresse fehlt, dann wird die Datei an die von ihr gewünschte Adresse geladen. Wenn man auch das Gerät wegläßt, wird die Geräteadresse »8« angenommen. Das Programm wird nach Ausführung dieses Befehls im Gegegensatz zum normalen LOAD-Befehl regulär beim nächsten Befehl weiterbearbeitet.

Beispiel:

```
10 print "jetzt wird geladen"
20 ←load "sprite",8,832
30 print "fertig mit laden"
```

2.5.2 Effiziente Ein- und Ausgabe

Das Commodore-Basic bietet dem Programmierer die Eingabebefehle GET # und INPUT # und den Ausgabebefehl PRINT #. Das reicht für die allermeisten Anwendungen aus. Allerdings weisen diese Befehle auch einige Nachteile auf: Zahlen werden bei PRINT # erst zu Strings umgewandelt und dann als Text abgespeichert, was viel Zeit und Platz verbraucht. Ebenso wird die Eingabe bei INPUT # verlangsamt. Bei jeder einzelnen Befehlsausführung wird zwischen den Ein- beziehungsweise Ausgabekanälen umgeschaltet. Dies wirkt sich besonders bei GET # sehr ungünstig auf die Geschwindigkeit aus. Es muß häufig mit den verhältnismäßig langsamen Strings gearbeitet werden, obwohl dies überhaupt nicht nötig wäre.

@BYTE, @INTEGER, @REAL, @WORD

←BYTE, ←INTEGER, ←REAL, ←WORD

Die Lösung bieten ein paar neue Befehle und Funktionen zur Ein-/Ausgabe. Die Befehle zur Ausgabe bestehen aus dem üblichen »←« und dem Namen eines numerischen Datentyps. Dem folgen dann eine oder mehrere mit Komma getrennte Ausdrücke (meist einzelne Variablen oder Zahlen). Das Ergebnis dieser Ausdrücke wird dann in den angegebenen Typ umgewandelt (falls nötig) und in den momentan aktiven Ausgabekanal geschrieben. »←BYTE 3« gibt zum Beispiel ein einzelnes Byte aus, das den Wert »3« hat. »←REAL 1.2345« gibt fünf Byte aus, die dem Speicherformat von 1.2345 entsprechen. Genau so arbeiten auch »←WORD X« und »←INTEGER X«, die genau zwei Byte schreiben. Die Eingabe wird im Gegensatz zur Ausgabe nicht über Befehle, sondern über Funktionen realisiert, da sie flexibler und effizienter angewandt werden können. Diese Funktionen beginnen grundsätzlich mit einem Klammeraffen, dem ähnlich wie bei den Befehlen ein numerischer Datentyp folgt. Es werden keine Parameter benötigt, weshalb man die Funktionen wie Variablen behandeln kann. Bei »A=@BYTE« wird ein Byte vom Eingabekanal geholt und der Variablen A zugewiesen. Bei

»PRINT @REAL« werden fünf Byte gelesen und als Fließkommazahl auf den Bildschirm ausgegeben. »POKE I,@BYTE« holt ein Byte und schreibt es nach Adresse I. Diese neuen Befehle und Funktionen sind in der Form allerdings nicht sonderlich nützlich, da sie nur auf das Standardausgabegerät Bildschirm schreiben und vom Standardeingabegerät Tastatur lesen können. Diesem Mißstand helfen folgende Befehle ab:

←OUT, ←IN, ←RESET

»←OUT n« richtet den Ausgabekanal auf die Datei n (genau so wie »CMD n«). »←IN n« richtet den Eingabekanal auf die Datei n. »←RESET« schaltet die Kanäle wieder auf die Standardgeräte Bildschirm und Tastatur. Wenn man zum Beispiel die 1000 Elemente des Real-Arrays »R(« in die Datei »TEST,S« schreiben will, so kann das folgendermaßen aussehen:

```
1000 open 1,8,2,"test,s,w"
1010 ←out 1
1020 for i=0 to 999: ←real r(i): next i
1050 ←reset
1060 close 1
```

Wenn eine Schreib- oder Leseoperation abgeschlossen ist, so muß unbedingt der »←RESET«-Befehl ausgeführt werden, damit man wieder Bildschirm oder Tastatur ansteuern kann. Denn auch die Befehle »GET«, »INPUT« und »PRINT« beziehen sich jeweils auf den aktuellen Ein-/Ausgabekanal. Trotzdem sollte man aus Zeitgründen möglichst selten umschalten. Das Betriebssystem läßt es nicht zu, daß Ein- *und* Ausgabekanal umgelenkt werden. Wenn der Eingabekanal umgelenkt wurde, muß also immer erst ein »←RESET« erfolgen, bevor der Ausgabekanal umgelenkt werden kann (oder umgekehrt). Man kann auch Daten von verschiedenem Typ in dieselbe Datei schreiben, wenn man peinlich genau darauf achtet, daß die Daten in exakt derselben Reihenfolge gelesen werden, wie sie geschrieben wurden. Auf der Programmdiskette finden Sie zwei Beispiele zu diesem Thema namens »BOSSDIR« und »RAMLOAD«.

2.5.3 Speicherverwaltung

←RAM, ←ROM und ←IOROM

Diese drei Befehle sind nur für den Spezialisten gedacht. Es ist am sinnvollsten, sie zusammen mit »£ALLRAM« zu verwenden. »←ROM« schaltet das ROM und den Input/Output-Bereich (Videoprozessor, SID, Farb-RAM und so weiter) ein, so daß darauf sehr schnell zugegriffen werden kann (mit POKE oder PEEK in Verbindung mit der Compilerdirektive »←RAM«, der ein »£« vorangeht). Dabei kann auf das RAM von $A000 bis $BFFF und $D000 bis $FFFF nicht mehr zugegriffen werden. Darum dürfen weder Programm noch Variablen in diesem Bereich liegen. »←IOROM« schaltet nur den Input/Output-Bereich und das Betriebssystem ein, nicht aber den Basic-Interpreter. Dann kann auf das RAM von $D000 bis $FFFF nicht mehr zugegriffen werden. »←RAM« schaltet wieder alle 64 Kbyte ein. Nach »←ROM« oder »←IOROM« sollten nur einfache Operationen mit Word oder Byte-Typen durchgeführt werden. Jede komplexere Operation ruft eine Unterroutine auf, die im allgemeinen wieder auf den RAM-Modus schaltet. Statt mit diesen Befehlen können Sie die Umschaltung übrigens auch gefahrlos selbst mit »POKE 1,X« durchführen (allerdings ebenfalls nur im ALLRAM-Modus).

←SEI und ←CLI

Auch diese Befehle sind eher für den professionelleren Programmierer gedacht. Sie sind mit den gleichnamigen Maschinensprachebefehlen identisch. Mit »←SEI« kann man den System-Interrupt sperren. Das ist sinnvoll, wenn man besonders zeitkritische Anwendungen programmiert, die keine kurzen Unterbrechungen vertragen. Allerdings wird nach »←SEI« die Tastatur nicht mehr abgefragt und die Zeit nicht hochgezählt. Darum sollte man den Interrupt mit »←CLI« sobald als möglich wieder einschalten. Außerdem kann man mit Hilfe dieser Befehle »←RAM«, »←ROM« und »←IOROM« auch dann sinnvoll einsetzen, wenn das Programm nicht im »£ALLRAM«-Modus läuft. Man sollte dann komplexere Operationen vermeiden. Sonst schalten eventuell aufgerufene Unterroutinen wieder auf den ROM-Modus zurück. Man sollte diese Befehle sehr sorgfältig handhaben und nach »←SEI« keine Ein-/Ausgabeoperationen durchführen.

2.5.4 Verarbeitung von Befehlen aus Basic-Erweiterungen

Das Commodore Basic V2 ist bekanntlicherweise recht spartanisch ausgestattet und sicher nicht für seinen Befehlsumfang berühmt. Dessen wird man sich sehr schnell bewußt, wenn man versucht, Grafik oder Töne zu programmieren. So ist es nicht verwunderlich, daß es eine Vielzahl von Erweiterungen und Zusatzroutinen gibt, die diesem Mangel begegnen. Aus der Sicht des Basic-Boss gibt es grundsätzlich drei verschiedene Sorten solcher Erweiterungen: Die simpelsten werden mit »SYS Adresse« aufgerufen. Eventuelle Parameter werden zuvor per POKE in bestimmte Speicherzellen gebracht (z.B. »POKE 900,A: SYS 49152«). Etwas benutzerfreundlicher ist die zweite Sorte, die ebenfalls per SYS aufgerufen wird, ihre Parameter aber an den SYS-Befehl angehängt erwartet (z.B. »SYS 49152, A, B$, C«). Die letzte Sorte ist die komfortabelste, denn sie arbeitet mit völlig neuen Befehlen (z.B. »LINE X1, Y1, X2, Y2«). Die simplen Erweiterungen machen dem Compiler keine Schwierigkeiten, da nur völlig normale Basic-Befehle verwendet werden. Genauso bereitwillig verarbeitet er die USR-Funktion. In beiden Fällen sind keine besonderen Vorkehrungen notwendig, vorausgesetzt, die Erweiterung benutzt keine wichtigen Teile des Speichers (dann muß man möglicherweise mit £HEAPEND arbeiten).

£OTHERON und £OTHEROFF

Schwierig wird es für den Compiler bei zusätzlichen SYS-Parametern oder gar völlig neuen Befehlen. Dann müssen spezielle Vorkehrungen getroffen werden, wozu man den Boss mit »£OTHERON« am Programmanfang veranlaßt. Wenn der Compiler nun einen unbekannten Befehl (genauer Token oder Token-Folge) findet, so nimmt er an, daß es sich um den Befehl einer Erweiterung handelt. Der Basic-Boss wertet den unbekannten Befehl samt Parameter und Trennzeichen aus und schreibt alles zusammen in einem Spezialformat ins Compilat. Die an den Befehl angehängten Parameter können voneinander mittels Komma oder irgendeinem anderen Sonderzeichen (z.B. mit einem Basic-Befehl) getrennt sein (z.B. »LINE X1,Y1 TO X2,Y2). Auch mehrere Trennzeichen oder gar keines werden akzeptiert. Die Reihenfolge und die Art der Trennzeichen muß aber von der Erweiterung erwartet werden, ebenso der Befehl selbst, da beim Compilat sonst ein »SYNTAX ERROR« auftritt. Denn das Compilat führt den

Befehl nicht selbst aus, sondern übergibt ihn dem Basic-Interpreter. Wenn sich eine Erweiterung im Speicher befindet, die sich ordnungsgemäß in den Interpreter eingebunden hat, dann wird sie den Befehl verarbeiten. Der Erweiterung können nur Strings und Real-Zahlen übergeben werden, da der Interpreter nur diese Typen kennt. Andere Datentypen werden daher nach Real umgewandelt. Vor allem ist zu beachten, daß die Erweiterung nur lesend auf die Variablen zugreifen darf. Eine Maschinenroutine zum Sortieren von Strings kann z. B. schon allein wegen der völlig neuen Stringverwaltung des Basic-Boss nicht verwendet werden. »£OTHERON« arbeitet übrigens auch im ALLRAM-Modus. Allerdings können dann die übergebenen Strings nicht länger als 80 Zeichen sein und man sollte beachten, daß die Erweiterung auch ihren Speicherplatz benötigt, den man ihr z. B. mit »£HEAPEND« sichern sollte. »£OTHEROFF« schaltet die Verwaltung von SYS-Parametern und Erweiterungsbefehlen ab. Dies ist der Normalzustand. Für Interessierte noch ein Wort zum Prinzip: Die Übergabe von SYS-Parametern ist eigentlich nicht ohne weiteres möglich, da die Variablenstruktur des Basic-Boss sich von der des Interpreters zu stark unterscheidet. Doch glücklicherweise bietet der Interpreter einen Vektor zum Holen eines arithmetischen Elements. Dieser Vektor wird bei »£OTHERON« vom Compilat auf eine eigene Routine gerichtet. Sie bereitet die Basic-Boss-Variablen auf und verfüttert sie an den Interpreter, der sie wiederum an das Maschinenprogramm der Erweiterung weitergibt. Die Trennzeichen liest die Erweiterung selbst. Bei neuen Befehlen kommt noch ein weiterer Vektor ins Spiel ($308/$309), über den das Compilat die Erweiterung anspringt.

£OTHER

Manche Erweiterungen arbeiten trotz »£OTHERON« mit dem Compilat nicht zusammen und erzeugen nur »SYNTAX ERROR«. Dies trifft auf eine bestimmte Klasse von Erweiterungen zu, die man oft daran erkennt, daß ihre Befehle mit einem Sonderzeichen eingeleitet werden, z. B. Pfeil links, Ausrufezeichen oder eckige Klammern oder ähnliches (es werden keine echten Tokens benutzt). Wenn dies nicht der Fall ist, dann sollten Sie ihr Basic-Programm einmal laden, ohne daß die Erweiterung im Speicher ist. Wenn die Erweiterungsbefehle nun immer noch völlig einwandfrei zu lesen sind, dann gehört Ihre Erweiterung auch zu dieser Gruppe. Deren Befehle kann der Basic-Boss nicht ohne weitere Hilfe verarbeiten, da er nicht weiß, wo

der Befehl aufhört und die Parameter anfangen. Wenn der Befehl z. B. »!COLA« lautet, dann kann das entweder ein einziger Befehl namens »!COLA« sein, oder auch »!COL A« meinen, mit A als Variable. Die »£OTHER«-Direktive schafft dem Abhilfe. Ihr sollte eine Liste aller im Programm verwendeter Befehle folgen, die voneinander per Komma getrennt sind, z. B. »£OTHER !COL, !PLOT, !HIRES«. Wenn eine Programmzeile nicht ausreicht, dann können auch mehrere Other-Direktiven eingesetzt werden. Sie sollten allerdings sicherheitshalber nicht in REM-Zeilen stehen, sondern per GOTO vom Interpreter ferngehalten werden (z. B. »10 GOTO11: £OTHER !COL«). Dann dürfen die Erweiterungsbefehle auch normale Basic-Befehle enthalten (z. B. »!TEXTON« enthält »ON«). »£OTHER« muß vor dem ersten Erweiterungsbefehl stehen (ansonsten ist der »7/CODE VERSCHOBEN«-Fehler möglich). Bei den moderneren Erweiterungen wird der OTHER-Befehl nicht benötigt (z. B. HiRes-Master aus »The Best of Grafik 3«).

£BOSSOFF und £BOSSON

In seltenen Fällen kann es vorkommen, daß ein Erweiterungsbefehl mit einem Basic-Boss-Befehl in Konflikt gerät. Wenn ein Erweiterungsbefehl z. B. »←LOAD« lautet, so wird der Basic-Boss fälschlicherweise annehmen, daß dies für ihn bestimmt ist. Darum kann man mit »£BOSSOFF« alle Direktiven und Befehle des Basic-Boss abschalten. Mit Ausnahme von »£BOSSON«, denn so schaltet man die Befehle wieder ein. Beide Direktiven können beliebig oft im Programm gebraucht werden. Welche Erweiterungen funktionieren ?
Meinen Erfahrungen nach arbeiten umfassende Basic-Erweiterungen wie Simons' Basic und G-Basic nicht mit dem Basic-Boss zusammen, da sie zu tief in die innere Struktur des Basic eingreifen. Die typischen Grafikerweiterungen wie Grafik 2000 (The Best of Grafik 2) oder der recht schnelle HiRes-Master (The Best of Grafik 3) arbeiten dagegen sehr gut mit dem Compilat zusammen. Bei HiRes-Master ist darauf zu achten, daß man die Strings mit »£HEAPEND $8000« in Sicherheit bringt. Beide Erweiterungen benötigen keinen Other-Befehl (nur »£OTHERON«). Sogar die Scroll-Machine (The Best of Grafik 2) kooperiert mit dem Boss. Allerdings wird hier die Other-Direktive benötigt. Andere Erweiterungen setzen teilweise den Basic-Start herauf, um so Platz für Sprites oder Grafik zu schaffen. Auch dies ist kein größeres Problem. Man muß dem Basic-Boss nur mit »£ROUTSTART Adresse«

mitteilen, wo der Basic-Start liegt (da Compilate, also Maschinenprogramme im Gegensatz zu Basic-Programmen nicht relokatibel sind). Bei geladener Erweiterung kann man die Startadresse mit »PRINT PEEK(43)+256*PEEK(44)« erfahren (die Endadresse für £HEAPEND mit »PRINT PEEK(55)+256*PEEK(56)«). Ansonsten ist noch darauf zu achten, daß die Erweiterung Werte von den Variablen holt und nicht in sie hineinschreibt. Man sollte beim Arbeiten mit Fremderweiterungen auch keine Fast-Variablen benutzen oder sicherstellen, daß die Speicherstellen $FB bis $FE ungeschoren bleiben. Wenn die Erweiterung Interrupts benutzt (z.B. Raster-Interrupts zum Bildschirmsplitting), dann sollte man keinesfalls »£ALLRAM« verwenden. Im Zweifelsfall hilft jedoch nur probieren. Das Compilat wird wie ein normales Basic-Programm geladen und gestartet. Natürlich muß die Erweiterung im Speicher sein. Ein Demoprogramm mit einer kleinen Beispielerweiterung befindet sich nebst Quellcode auf der Diskette unter dem Namen »OTHER«.

2.6 Änderungen beim Commodore Basic V2

Aufgrund des verschiedenen Arbeitsprinzips von Interpreter und Compiler ergeben sich bei einigen der normalen Befehle Unterschiede oder neue Eigenschaften. Außerdem habe ich ein paar Befehle verbessert. All das ist im folgenden aufgeführt.

2.6.1 Verbesserungen

An ein paar Stellen habe ich Verbesserungen oder Korrekturen vorgenommen. Diese erweitern die Möglichkeiten, führen aber nicht dazu, daß Inkompatibilitäten des Compilers entstehen. Allerdings funktioniert ein Basic-Programm teilweise nicht mehr ordnungsgemäß unter dem Interpreter, wenn diese Möglichkeiten genutzt werden.

Hexadezimale Zahlen

Es ist nun erlaubt, Zahlen auch in hexadezimaler Schreibweise anzugeben. Eine hexadezimale Zahl beginnt mit »$«, dem dann beliebig viele Ziffern (0-9 und A-F) folgen können. Auf diese Weise können aber nur ganze Zahlen angegeben werden.

A=ASC(A$)

Die ASC-Funktion wurde leicht geändert. Wenn A$ leer ist, dann liefert ASC den (Byte-)Wert 0 und es erfolgt kein »illegal quantity error«. Dies ist sinnvoll, weil bei »GET A$« ein Nullbyte zu einem Leerstring gemacht wird.

A=FRE(X)

Diese Funktion lieferte beim Interpreter die Größe des noch freien Speicherbereichs. Das Ergebnis war gelegentlich negativ (wenn der Bereich größer als 32 Kbyte war). Nun liefert sie grundsätzlich die Größe des noch freien Stringspeicher, und zwar positiv.

RESTORE

kann nun auch eine Zeilennummer als Parameter haben. Wenn sich der nächste READ-Befehl zum Beispiel auf die Zeile 150 beziehen soll, so kann man nun »RESTORE 150« schreiben. Damit muß man die Daten nicht mehr unbedingt von vorne nach hinten lesen und braucht nicht mehr so peinlich genau auf die Reihenfolge zu achten.

READ und DATA

An der Syntax ändert sich nichts. Trotzdem sind diese Befehle ein gutes Stück leistungsfähiger geworden. Die Compilerdirektive »£DATATYPE Typ« ermöglicht es, die Daten bei DATA bereits in ihrem Bestimmungstyp abzulegen, was Speicher spart und die Verarbeitung wesentlich beschleunigt. Näheres können Sie unter 2.4.1 »Steuerung der Datentypen« bei »£DATATYPE« nachlesen.

FOR und NEXT

Es existiert eine zweite Version der FOR-NEXT-Schleife, die mit »£FASTFOR« aktiviert werden kann. Näheres hierzu unter 2.4.2 »Optimierungsmöglichkeiten« bei »FASTFOR«.

2.6.2 Befehle mit Sinnverlust

Einige Basic-Befehle sind hauptsächlich für den Direktmodus gedacht, weshalb gewisse Schwierigkeiten bei deren Realisierung im Compilat auftreten.

LIST, NEW, STOP

Alle drei Befehle sind für den Basic-Boss vollkommen identisch mit END. Denn der LIST-Befehl macht im Compilat wenig Sinn, da er bestenfalls die SYS-Zeile zeigen und das Programm anschließend sowieso beenden würde. NEW ist im Programm ziemlich bedeutungslos und eine richtige Implementation von STOP scheint mir auch nicht sinnvoll.

CONT

ist im Programm eigentlich völlig sinnlos. Meinen Erfahrungen nach wirkt es im Basic-Programm wie eine Endlosschleife. Darum produziert auch der Basic-Boss beim Auftreten von CONT eine solche.

LOAD, SAVE, VERIFY

Sie funktionieren genauso wie unter dem Interpreter. LOAD lädt Daten in den Speicher und springt an den Programmanfang. SAVE speichert das aktuelle Programm im Basic-Speicher ab und VERIFY vergleicht es mit einem anderen Programm auf Diskette (und erzeugt gegebenenfalls einen »verify error«). SAVE und VERIFY wurden nur der Vollständigkeit halber implementiert.

2.6.3 Änderungen

Hier sind die Befehle aufgeführt, die im Compilat nicht ganz so reagieren, wie unter dem Interpreter.

INPUT

Beim Konzept des INPUT-Befehls ist eine Kleinigkeit verändert: Bei »INPUT A,B,C« kann wie sonst auch ein »redo from start«-Hinweis auftauchen, wenn die Eingabe unsinnig ist. Beim Interpreter muß der Benutzer dann alle drei Werte von neuem eingeben. Beim Basic-Boss muß er die Werte ab dem Parameter eingeben, bei dem der Fehler auftrat. Die vollkommene Kompatibilität erschien mir ziemlich sinnlos.

IF A$ THEN

Die Bedingung wird im Compilat als erfüllt betrachtet, wenn »A$« nicht leer ist. Der Interpreter reagiert hier scheinbar nach Lust und Laune (hin und wieder auch mit einem »formula too complex error«) was mich vermuten läßt, daß hier ein Fehler im Interpreter vorliegt. Das einst in der Zeitschrift 64'er veröffentlichte Programm »TURBODIR« arbeitet als Compilat nicht, da es den Betriebssystemfehler ausnutzt. Es wird dort die falsche Variable auf die Länge 0 überprüft. Da der Interpreter nicht die angegebene Variable prüft, sondern die andere, funktioniert es unter dem Interpreter trotzdem.

2.6.4 Problemfälle

GOSUB...RETURN und FOR...NEXT

Sie funktionieren im Normalfall wie erwartet, doch in einer bestimmten Situation stürzt »RETURN« ab:

```
10 gosub 20: end
20 for i=1 to 10
```

```
30 goto 50
40 next i
50 return
```

Diese Befehlsfolge funktioniert beim Interpreter einwandfrei. Das Compilat dagegen stürzt ab. Der Grund dafür ist darin zu suchen, daß der Basic-Boss GOSUB und RETURN in die Maschinensprachebefehle »JSR X« und »RTS« übersetzt, was auch als das einzig sinnvolle erscheint. Diese Befehle beanspruchen aber den Stack. Da jedoch FOR seine Parameter auch auf den Stack legt, verarbeitet RETURN die Werte, die eigentlich für NEXT gedacht waren und springt irgendwo hin. Dies könnte man mit einer speziellen Kennung der Stack-Inhalte vermeiden, ähnlich wie der Interpreter das macht. Das würde die Geschwindigkeit aber drastisch verringern. Eine solche Stack-Verwaltung ist dennoch in Planung (zuschaltbar). Es gibt übrigens keinen »return without gosub error«, da ein überflüssiges »RTS« aus dem Maschinenprogramm wieder ins Basic zurückspringt. Auch dies wäre mit einer Kennung nicht der Fall.

DIM

Beim Gebrauch des DIM-Befehls gibt es nur wenig Einschränkungen. Allerdings muß man sich darüber im klaren sein, daß der DIM-Befehl eigentlich eine Compilerdirektive ist, weshalb er bereits während des Kompilierens ausgewertet wird. Darum ist es nicht erlaubt, Variablen beim Dimensionieren von Feldern zu benutzen (zum Beispiel »DIM A(B)«). Außerdem sollte eine Variable vor ihrer Benutzung mit DIM dimensioniert sein (auch wenn sie nur elf Elemente enthalten soll), da dann teilweise ein effizienterer Code erzeugt werden kann (X-indizierte Adressierungsart). Aus der Direktiven-Natur von DIM resultiert auch das Verbot folgender Konstruktion, da sich der Compiler nicht an der tatsächlichen Abarbeitungsreihenfolge orientiert, sondern die Programme von vorne nach hinten durchgeht:

```
10 dim a(10)
20 goto 40
30 dim a(20)
40 :
```

DEF FN

Da der Compiler im Gegensatz zum Interpreter die Ausdrücke nicht während der Programmausführung auswertet, ergeben sich auch hier Probleme. Eine Funktion muß vor ihrer Benutzung definiert werden (das heißt in einer Zeile mit kleinerer Nummer). Eine FN-Funktion darf darüber hinaus nicht mehrmals definiert werden, wie das in Basic möglich ist. Eine Schachtelung von FN-Ausdrücken ist erlaubt, ebenso eine Mischung mit Variablen des Typs »Constant«. Man sollte dennoch sehr sparsam mit DEF FN umgehen, da es das Compilat stark verlängern kann, denn beim Auftreten einer FN-Funktion wird sie jedesmal neu ausgewertet. Dies ist in der Natur von FN begründet: Es handelt sich um ein Makro und nicht um ein Unterprogramm.

LOAD

Unter dem Interpreter kann ein Basic-Programm andere Basic-Programme nachladen. Das nachgeladene Programm belegt dann den Speicher seines Vorgängers und kann auf seine Variablen zugreifen. Dies funktioniert aber nur, wenn das nachgeladene Programm kleiner als sein Vorgänger ist und die Variablen keine String-Variablen sind, die in der Form »A$="..."« gesetzt wurden. Ein Basic-Boss-Compilat kann auch ein beliebiges anderes Programm nachladen. Der Nachfolger kann aber nicht auf die Variablen seines Vorgängers zugreifen. Zur Not kann man Werte mit POKE, PEEK oder mit Adreßvariablen übergeben (zu empfehlen ist der Bereich von 828 bis 1023).

PRINT A, IF A THEN und STR$(A)

Bei allen drei Befehlen kann es in seltenen Fällen zu Problemen mit den Datentypen des Basic-Boss kommen. Denn bei diesen drei Befehlen ist der Mindesttyp Boolean, also der niedrigste Typ überhaupt. Wenn »A« ein Ausdruck ist, dann wird dieser Ausdruck in seinem Typ berechnet. Die Befehlsfolge »W1=5: W2=2: PRINT W1/W2« wird zum Beispiel nicht »2.5«, sondern »2« ausgeben, falls »W1« und »W2« Word-Variablen sind. Dessen sollte man sich bewußt sein, wenn man diese Befehle benutzt. Ändern kann man das mit den Direktiven »PRINTTYPE« und »BOOLEANTYPE«.

2.7 Speicherplatz und Geschwindigkeit

Die Länge des Programms schwankt etwa zwischen 80% und 170% der Länge des ursprünglichen Programms. Man kann sicher auch Faktoren von 10% oder 1000% erreichen, wenn man es darauf anlegt. Normalerweise wird dies jedoch nie der Fall sein. Wenn man effizient programmiert, schwankt der Faktor zwischen 80% und 120%. Dazu kommen noch die angelinkten Routinen (Stringverwaltung und all das, was normalerweise im ROM steht). Der Variablenbereich wird immer kürzer. Eine als Byte deklarierte Variable benötigt nur ein Byte. Der Interpreter benötigt dagegen grundsätzlich sieben Byte pro Variable. Der Geschwindigkeitsfaktor schwankt zwischen drei und 1000. Bei Verwendung von Word und Byte beträgt er etwa 50 bis 300. Bei Real liegt er zwischen zwei und acht und bei String zwischen fünf und zehn. Größenordnungsmäßig entspricht die Geschwindigkeit der des vor einiger Zeit im 64'er-Magazin veröffentlichten As-Compilers, wobei das Compilat des Basic-Boss aber noch etwas effizienter ist.

2.8 Effiziente Programme

Hier wird beschrieben, wie sie kurze und schnelle Compilate erhalten können. Das macht der Basic-Boss zwar ziemlich einfach, doch man muß auch selbst seinen Teil dazu tun. Wenn man kurze und schnelle Compilate erhalten will, sollte man folgende Grundregeln beachten:

2.8.1 Datentypen

2.8.1.1 Der richtige Typ

Soweit es irgendwie möglich ist, muß der Datentyp Byte verwendet werden, denn er ist der eigentliche Datentyp des C64 und darum der schnellste und kürzeste. Der Wertebereich von 0 bis 255 reicht für viele Anwendungen aus (zum Beispiel Stringverarbeitung). Falls ein größerer Wertebereich wünschenswert ist, sollte man Word-Variablen verwenden. Sie sind halb

so schnell und verbrauchen doppelt so viel Platz im Programm- und im Variablenspeicher. Nach Möglichkeit sollte man außerdem FAST-Variablen benutzen (vor allem bei Adressen bei POKE- und PEEK-Adressen). Wenn man unbedingt mit vorzeichenbehafteten Zahlen rechnen will, weil man auf die Operatoren »>«, »«<, »>=« und »<=« nicht verzichten kann, dann sollte man den Integer-Typ dem Real-Typ in jedem Falle vorziehen, soweit das möglich ist, denn Integer ist in den meisten Fällen so schnell wie Word, außer bei »>«, »«<, »>=« und »<=«. Wenn diese Operatoren nicht benötigt werden, kann man auch bei Byte und Word mit vorzeichenbehafteten Zahlen rechnen, wobei man aber bei Umwandlungen und der Division aufpassen muß. Nur wenn es unvermeidbar ist, sollten Sie Real-Variablen oder -Funktionen verwenden. Ebenso sollte man Strings so weit als möglich vermeiden. Allerdings benötigen Operationen mit Strings oder Real-Zahlen aufgrund eines äußerst sparsamen Parameterübergabeprinzips weniger Programmspeicher als die meisten Operationen mit Word und Integer (zum Beispiel Addition: 17 bis 19 Byte bei Word, 9 Byte bei Real).

2.8.1.2 Umwandlungen

Es ist äußerst wichtig, unnötige Umwandlungen zu vermeiden. Man sollte bei Befehlen oder Operationen grundsätzlich die Typen benutzen, die erwartet werden. Welche das sind, läßt sich bei 2.3 »Datentypen« unter »Typen bei Funktionen und Befehlen« nachlesen. Der Befehl »POKE A,B« ist zum Beispiel grauenhaft ineffizient, wenn »A« und »B« vom Typ Real sind. Dann wird eine Operation (POKE) zu drei Operationen gemacht (A → Word, B → Byte, POKE) und die Geschwindigkeit stark reduziert. Ähnlich ist es bei den anderen Befehlen. Es gibt aber auch ein paar Umwandlungen, die beliebig verwendet werden dürfen, weil sie weder Speicherplatz noch Zeit benötigen. Dies sind: Boolean ↔ Byte, Word ↔ Integer, Integer → Byte und Word → Byte. Nur wenig Platz und Zeit benötigen folgende Umwandlungen: Byte → Word, Byte → Integer. Vor allem sind Umwandlungen von und nach Real zu vermeiden. Man sollte den Datentyp für eine Variable auch anhand deren Verwendung entscheiden. Wenn zum Beispiel die Variable »A« nur die Werte 0 und 1 annimmt, sollte man sie trotzdem nicht als Byte definieren, sondern als Real, falls sie ausschließlich mit Real-Variablen verrechnet werden soll.

2.8.2 Konstanten, Operationen, Befehle, Arrays

Konstanten

Die Verwendung von konstanten Zahlen anstatt von Variablen beschleunigt die Verarbeitung bei Byte, Word und Integer und spart Speicherplatz. Vor allem bei Adressen für POKE und PEEK sollte man Konstanten verwenden (oder FAST-Variablen). Wenn Sie aber keine nichtsagenden Zahlen verwenden wollen, so hilft Ihnen der Datentyp »Constant«.

Operationen

Wenn Sie mit Word, Integer und Byte arbeiten, sollten Sie möglichst nur einfache Operationen verwenden (Addition, Subtraktion, Zuweisung, Vergleiche, AND, OR). Operationen wie die Multiplikation, Division oder gar Potenzierung sollten vermieden werden, wobei aber für Multiplikation und Division spezielle Routinen für Word und Integer benutzt werden, die noch relativ schnell sind. Unbedingt zu vermeiden sind folgende Funktionen: SIN, COS, TAN, ATN, EXP, LOG, SQR, RND und vor allem Potenzierung. Wie das häufig realisiert werden kann, wird später bei 2.8.4 »Tabellen« beschrieben.

Befehle

Die Befehle wurden größtenteils stark beschleunigt. Dies gilt in besonderem Maße für:

POKE, PEEK, GOTO, GOSUB, RETURN, FOR...NEXT (FASTFOR), LET, ON, WAIT.

Sogar der PRINT-Befehl ist bei Word, Integer und Byte um einiges schneller, da spezielle Ausgaberoutinen für diese Typen bestehen.

Arrays

Vermeiden Sie mehrdimensionale Arrays, denn diese sind sehr langsam, da zu deren Berechnung Multiplikationen notwendig sind und zwar genau »(Dimensionen-1) Stück«. Vermeiden

Sie auch Arrays vom Typ Real und String und verwenden Sie »£SLOWARRAY« nur zum Austesten. Bei eindimensionalen Arrays brauchen Sie sich bei den Datentypen Integer, Word oder Byte keine Sorgen hinsichtlich der Geschwindigkeit zu machen, da der Zugriff auf diese Arrays aufgrund spezieller Algorithmen sehr schnell erfolgt. Dimensionieren Sie Ihre Felder immer ganz am Anfang des Programms, auch wenn Sie nicht mehr als elf Elemente benötigen (DIM A(10)). Denn um eine besonders effiziente Adressierungsart des Prozessors 6510 anwenden zu können, muß der Basic-Boss von Anfang an wissen, wie groß ein Array ist. Dies gilt aber nur für eindimensionale Arrays vom Typ Word, Integer oder Byte, die weniger als 128 oder 256 Elemente beherbergen. Falls Sie mehrdimensionale Arrays verwenden, sollten Sie bei den hinteren Dimensionen Konstanten verwenden, soweit das möglich ist. Sie sollten Ihre Arrays also so anlegen, daß Sie statt »A (10,I)« die Form »A (I,10)« verwenden. Damit sparen Sie eine Multiplikation und viel Zeit, weil der Basic-Boss das Array teilweise selbst berechnet und das Compilat nur den Rest berechnen muß. Bei »A (I,10,15)« werden so zwei Multiplikationen gespart.

2.8.3 Vermeidung überflüssiger Befehlsausführung

Sie sollten Ihre Programme so schreiben, daß nur die Befehle ausgeführt werden, die nötig sind. Insbesondere sollte man die unnötige Verarbeitung von Realzahlen oder Strings vermeiden. Folgendes Programm kann man zum Beispiel wesentlich beschleunigen:

```
10 get a$
20 if a$="a" then ...
30 if a$="b" then ...
40 ...
50 rem hier gehts weiter
```

indem man folgende Zeile ändert:

```
10 get a$:if a$="" then 50
```

Noch einmal wesentlich schneller geht es so:

```
5 if peek (198)=0 then 50
10 get a$
```

Hier wird überprüft, ob der Tastaturpuffer leer ist. Wenn ja, wird überhaupt keine Stringoperation durchgeführt. Dies ist die optimale Lösung.

2.8.4 Tabellen

Stark beschleunigen kann man seine Programme mit Hilfe von Tabellen. Betrachten Sie zum Beispiel einmal folgende Routine zum Setzen eines Punktes auf den Bildschirm. »X« (0 bis 39) und »Y« (0 bis 24) enthalten die Position und die Routine wird mit »GOSUB 1000« aufgerufen. Alle Variablen seien vom Typ Word (weil sie mit Word verrechnet werden).

```
1000 poke 1024+40*y+x,160: return
```

Wenn diese kleine Routine häufig aufgerufen wird, dann wird das Programm ziemlich langsam, da die Multiplikation verwendet wurde. Die Additionen fallen kaum ins Gewicht. Es geht aber auch besser. Man kann sich die Multiplikation sparen, wenn man alle möglichen Multiplikationen schon vorher ausführt und dann nur noch die Ergebnisse benutzt. Das geht so:

```
10 dim sc(24)
20 gosub 2000
30 ...
1000 poke sc(y)+x,160: return
2000 for y=0 to 24: sc(y)=1024+y*40: next y: return
```

Nun werden die Multiplikationsergebnisse am Programmbeginn in ein Array eingelesen, weshalb im Programm direkt auf sie zugegriffen werden kann. Ähnliches kann man zum Beispiel auch bei Sinuswerten oder bei den oft benötigten Zweierpotenzen machen, denn Sinusberechnung und Potenzierung sind noch langsamer.

2.8.5 Verkürzung von Berechnungen

Sehen Sie sich folgende Programmzeile an:

```
10 a=10+20
```

Wenn man den Interpreter mit »RUN« startet, so führt dieser eine unnötige Addition aus. Denn statt »10+20« wäre es doch sinnvoller, einfach »30« zu schreiben. Wenn man die Zeile kompiliert und das Compilat untersucht, wird man feststellen, daß dieses keine Addition durchführt. Der Basic-Boss hat also erkannt, daß er hier selbst etwas vereinfachen kann und bereits während dem Kompilieren »10« und »20« zusammengezählt, weshalb das Compilat für »A=10+20« oder »A=30« exakt dasselbe ist. Diese Sofort-Berechnungen kann der Compiler mit den Grundrechenarten »+«, »–«, »*« und »/« durchführen. Außerdem kann er potenzieren oder negieren (zum Beispiel –(1+2)). Alle diese Berechnungen werden grundsätzlich im Real-Typ durchgeführt. Auch beim Ausdruck »1+2+A« kann der Compiler vereinfachen. Bei »A+1+2« hat er aber Schwierigkeiten. Darum sollte man »A+(1+2)« schreiben, wenn man effizienten Code haben will.

2.8.6 £FASTFOR, £DATATYPE, £SHORTIF

Soweit es möglich ist, sollte FASTFOR verwendet werden (siehe unter 2.4.2 »Optimierungsmöglichkeiten« und »£FASTFOR«). Bei Verwendung von DATA und READ sollten Sie mit »£DATATYPE Typ« unbedingt den Typ der Daten spezifizieren (näheres bei »£DATATYPE«). Außerdem ist »£SHORTIF« zu empfehlen.

2.9 Anpassung vorhandener Programme

Vermutlich besitzen Sie Basic-Programme, die Sie zu einer Zeit schrieben, als Sie vom Basic-Boss noch nichts wußten. Wenn Sie ein solches Programm an den Basic-Boss anpassen wollen, dann ergeben sich eine Reihe charakteristischer Probleme. Wenn Sie diese meistern wollen,

dann sollten Sie folgende Schritte beachten: Achten Sie auf DIM und DEF FN und stellen Sie fest, ob diese Befehle in unzulässiger Weise benutzt werden. DEF FN muß am Programmanfang stehen (vor der Verwendung der Funktion) und bei DIM dürfen keine Variablen zur Definition von Arrays verwendet werden (siehe »Änderungen beim Commodore-Basic V2« unter 2.4.6 »Problemfälle«).
Stellen Sie dann fest, ob das Programm irgendeinen Speicher belegt, zum Beispiel für nachzuladende Maschinenprogramme oder für einen veränderten Zeichensatz oder ähnliches. Ein sicheres Merkmal hierfür ist ein POKE-Befehl, der die Adresse 56 (Basic-Speicherende) zum Ziel hat. Meist folgt diesem Befehl der CLR-Befehl. Den POKE-Befehl sollten Sie dann entfernen und stattdessen die Compilerdirektive »£HEAPEND Adresse« benutzen, die dem Basic-Boss das Basic-Speicherende direkt angibt. Mit ihr können Sie sicherstellen, daß das Compilat den reservierten Bereich unangetastet läßt. Sie sollten auch alle anderen POKE-Befehle entfernen, die die Adressen 43 bis 56 betreffen. Falls der dem Compilat verbleibende Speicherbereich nicht ausreicht (der Compiler meldet »heapend unter heapstart«), dann müssen Sie entweder den reservierten Bereich verlegen oder einen Teil des Compilats (Programmbereich, Variablenbereich oder Stringbereich) an einen anderen Ort legen (möglicherweise wird »£ALLRAM« nötig). Auf jeden Fall sollten Sie sicherstellen, daß kein POKE-Befehl im Programm auf einen der Bereiche des Compilats zugreift.
Wenn das Basic-Programm sehr lang ist, können Sie vorsorglich »£ALLRAM« verwenden. Dabei ist allerdings darauf zu achten, daß dann der gesamte Speicher benutzt wird. Wenn zum Beispiel ein Maschinenprogramm ab $C000 (49152) liegt, dann sollten Sie das Basic-Speicherende mit »£HEAPEND $C000« vor diesen Bereich legen. Wenn sich SYS-Befehle mit angehängten Parametern im Programm befinden, dann wird »£OTHERON« nötig. Dabei ist allerdings sicherzustellen, daß die Maschinenroutinen nicht versuchen, die übergebenen Variablen zu verändern, sondern nur deren Wert verarbeiten. Auch vertragen sich nicht alle Maschinenprogramme mit dem Basic-Boss. Hier hilft am besten Ausprobieren. Im Gegensatz zu anderen Sprachen gibt Basic nicht gerade Hilfestellung zum strukturiertem Programmieren, dies wird eher von den unbequemen Zeilennummern und dem Konzept der globalen, zweistelligen Variablen noch erschwert. Darüber hinaus halten es viele Basic-Programmierer mit der Ordnung nicht allzu genau, weshalb ein Basic-Programm leicht in ein unkontrollierbares Chaos ausarten kann. Darum ist bei der Anpassung im besonderen folgendes zu beachten:

Ein Basic-Programm arbeitet im Normalfall ausschließlich im Realtyp. Da aber das Compilat bei Berechnungen in diesem Typ nicht übermäßig schnell ist, würden Sie sicher gerne auch die restlichen Typen des Basic-Boss nutzen. Dies sollte man aber sehr sparsam tun, da bei einem Basic-Programm nicht so leicht festzustellen ist, wo welche Variablen benutzt werden. Darum ist es anzuraten, nur einige wenige Variablen umzudefinieren, die an zentralen und für die Geschwindigkeit besonders bedeutsamen Orten benutzt werden. Man muß genau wissen, wo diese Variablen überall gebraucht werden, damit sichergestellt ist, daß sie nicht andernorts vom Typ Real sein müssen, während man sie als Word, Integer oder Byte deklariert. Der Code wird auf diese Weise zwar recht lang, doch das läßt sich meist verkraften (»£ALLRAM«). Für diese Arbeit kann ein Crossreference-Programm sehr nützlich sein, das genau anzeigt, wo welche Variablen benutzt werden.
Hin und wieder stellt auch die FOR-NEXT-Schleife ein Problem dar. Wenn in einem Unterprogramm aus einer Schleife mit GOTO herausgesprungen wird, dann riskieren Sie in Basic nur einen Stacküberlauf, das Compilat stürzt dagegen beim nächsten RETURN ab (siehe unter »Änderungen beim Commodore Basic V2« unter 2.6.4 »Problemfälle«). Verhindern können Sie dies, indem Sie nicht mit einem bloßen GOTO aus der Schleife springen, sondern die Schleifenvariable zunächst auf den Endwert setzen, einen NEXT-Befehl ausführen, der dann nicht wieder zum Anfang der Schleife führt, weshalb Sie nun unbehindert mit GOTO arbeiten können. Zu diesem Thema finden Sie ein Beispielprogramm namens »FORRETURN« auf der Basic-Boss-Diskette.

2.10 Es funktioniert nicht

Wenn Ihr Programm Schwierigkeiten macht, dann sind folgende Gründe denkbar:

- Das Programm ist in sich fehlerhaft
- Die Datentypen werden falsch angewandt
- Das Programm verträgt die Eigenheiten des Compilers nicht

Wenn die erste Möglichkeit zutrifft, kann ich Ihnen nicht weiterhelfen. Dann müssen Sie nach den allgemeinen Regeln für das Ausmerzen von Fehlern verfahren. Wenn Sie aber den Verdacht

haben, die Datentypen falsch anzuwenden, können Sie versuchsweise alle Variablen auf Real setzen, indem Sie sämtliche Deklarationen unwirksam machen (mit »£IGNORE« und »£USE« oder indem Sie den »@« hinter den REM-Befehlen entfernen. Das Compilat wird dann zwar ziemlich lang und ineffizient, wenn aber der Speicher ausreicht (zur Not kann man »£ALLRAM« zu Hilfe nehmen) und das Programm dann läuft, so ist dies ein Hinweis darauf, daß irgendeine Variable vom Typ Real sein sollte, dies aber nicht ist. Nun können Sie den Fehler suchen, indem Sie entweder aus den Reaktionen des Programms schlußfolgern oder die Variablen grüppchenweise wieder auf den Wunschtyp umdefinieren. Wenn das Programm dann auch nicht läuft, dürfte wohl eine Eigenheit des Compilers daran schuld sein. Wahrscheinlich wird aus einer FOR-NEXT-Schleife gesprungen, weshalb das Compilat beim darauffolgenden RETURN abstürzt. Möglich ist auch, daß das Basic-Programm Maschinenroutinen aufruft, die sich mit dem Compilat nicht vertragen. Dabei ist eine Überlappung von Speicherbereichen denkbar, da das Compilat häufig länger ist als das Original. Dazu sollten Sie sich die 2.6.4-»Problemfälle« unter »Änderungen beim Commodore-Basic V2« ansehen. Schließlich kommt noch ein Fehler im Basic-Boss in Betracht. Ein solcher ist durchaus möglich, weil der Basic-Boss noch sehr jung ist. In einem solchen Falle sollten Sie sicherstellen, daß tatsächlich ein Compilerfehler vorliegt. Das Programm sollte also unter dem Interpreter einwandfrei laufen und die oben beschriebenen Fehler sollten weitgehend ausgeschlossen sein. Dann wäre ich Ihnen überaus dankbar, wenn Sie den Fehler so weit als möglich isolieren könnten, das heißt, das fehlerhafte Programm sollte möglichst kleingemacht werden, ohne daß der Fehler verschwindet. Anschließend sollten Sie mir das fehlerhafte Programm mit möglichst *genauer* Fehlerbeschreibung (Auswirkungen des Fehlers) zuschicken; sehr, sehr kleine Programme als Ausdruck und alles, was darüber hinausgeht, auf Diskette. Dabei dürfen Sie die Versionsnummer Ihres Compilers auf keinen Fall vergessen.

2.11 Rechtliches

Das Compilat enthält von mir verfaßte Routinen und Algorithmen, vor allem im Routinenbereich. Trotzdem beanspruche ich kein Urheberrecht auf das Compilat, sofern diese Routinen nicht ausgebaut und für andere Zwecke verwendet werden. Das Compilat kann also unein-

geschränkt kopiert oder verkauft werden, soweit der Autor des kompilierten Basic-Programms dies zuläßt. Bei ihm allein liegt das Urheberrecht für das kompilierte Programm. Allerdings halte ich es für fair, wenn im Programm oder der Anleitung erwähnt wird, daß es mit dem Basic-Boss kompiliert wurde. Für den Compiler selbst gilt das volle Urheberrecht. Ebenso gilt es für die Compileranleitung. Beides darf nicht (auch nicht teilweise) ohne schriftliche Genehmigung kopiert oder in irgendeiner Weise vervielfältigt werden. Für vom Compiler verursachte Schäden wird übrigens keine Haftung oder juristische Verantwortung übernommen.

2.12 Fehlermeldungen

Es gibt zwei Sorten von Fehlermeldungen. Die einen meldet der Basic-Boss bereits während des Kompilierens und die anderen erscheinen erst bei der Ausführung des Compilats.

2.12.1 Die Meldungen des Basic-Boss

Diese Meldungen erscheinen während des Kompiliervorganges. Da es sich bei dem Basic-Boss um einen Zwei-Pass-Compiler handelt, tauchen die meisten Fehlermeldungen doppelt auf (einmal in Pass 1 und einmal in Pass 2). Die Fehlermeldung besteht aus mindestens zwei Zeilen. In der ersten Zeile werden Zeilennummer und dahinter eine Art Extrakt der Zeile dargestellt. Dieser Extrakt besteht aus den verwendeten Rechenoperatoren, einem Punkt für jeden Befehl und den Doppelpunkten, die die Befehle trennen. Der Extrakt erlaubt eine genauere Lokalisation des Fehlers. Zumindest die Doppelpunkte geben einen genauen Aufschluß über die Position des Fehlers und auch die Operatoren können nützlich sein. Wenn gelegentlich unbekannte Operatoren auftauchen (z. B. »k,g,v,u,...«), so sollten Sie sich nicht weiter darum kümmern, da dies einer internen Darstellung im Basic-Boss entspringt. Die Zeile darunter enthält die Fehlernummer gefolgt von der eigentlichen Fehlermeldung. Die Fehlernummer erleichtert das Auffinden in der Fehlerliste. Es gibt auch einige Fehler, die nie erscheinen dürfen und deren Text nur für mich einen Sinn ergibt. Falls nämlich keine Fehlernummer angezeigt wird und der Fehler mit einem Pfund-Zeichen beginnt, dann zeigt das ein Problem im Basic-Boss an,

das eigentlich nicht existieren dürfte. In solchen Fällen kann eine Aufteilung eines komplizierten Ausdrucks in mehrere kleinere Ausdrücke das Problem beheben. Ein Pfund-Fehler kann auch auftauchen, wenn bestimmte Fehler vorausgegangen sind, die den Basic-Boss unvermittelt aus der Befehlsbearbeitung herausgerissen haben. In diesen Fällen löst sich das Problem mit der Behebung der anderen Fehler von selbst. Man sollte sich beim Auftreten eines Fehlers nicht auf die angegebene Fehlerursache versteifen, da häufig verschiedenartige Schreibfehler zum gleichen Fehler führen. Darum sollte man bei unauffindbaren Schwierigkeiten die allgemeine Syntax an der Fehlerstelle überprüfen. Insbesonders den vorangehenden Ausdruck sollte man genau untersuchen, da in ihm ein Fehler stecken kann, der den Compiler veranlaßt, das Ausdrucksende zu früh zu vermuten, was zu einer Vielzahl von Fehlermeldungen führen kann. Bei längeren Variablennamen sollte man sich davon vergewissern, daß sie keine Basic-Befehle enthalten, da auch dies vielfältige Auswirkungen haben kann. Wenn ein Fehler angezeigt wurde, kann ich für die Funktionstüchtigkeit des Compilats nicht mehr garantieren. Einige Fehler sind so schwerwiegend, daß der Compiliervorgang bei ihrem Auftreten sowieso abgebrochen wird. Wenn als Fehlerzeile 65 535 angezeigt wird, dann trat der Fehler nach Bearbeitung der letzten Programmzeile auf.

Die Fehlerliste

Die Fehler sind nach ihrer Nummer geordnet. Diese Ordnung erhebt keinen Anspruch darauf, sinnvoll zu sein, da die Liste nur zum Nachschlagen gedacht ist. Dem Fehler folgen eine nähere Beschreibung des Problems und eventuell Vorschläge zu dessen Behebung.

1/HEAPEND IST UNTER HEAPSTART

Das Ende des Stringspeichers liegt unter dessen Anfang. Das ist typischerweise dann der Fall, wenn das Compilat zu lang wird und nicht mehr in den Basic-Speicher paßt, weshalb es über das Stringspeicherende (normalerweise 40960 bzw. $A000) hinausragt. Abhilfe schafft hier z. B. der »£ALLRAM«-Befehl und/oder das Verlegen des Stringspeichers mit »£HEAPSTART« und »£HEAPEND«. Der Fehler tritt auch dann auf, wenn Sie das Compilat in einen anderen Speicherbereich legen, z. B. nach $C000. Dann muß ebenfalls »£HEAPEND« angewandt werden.

2/DIVISION DURCH NULL

Es wird eine Berechnung mit direkten Zahlen ausgeführt, die eine Teilung durch »0« verlangt (z. B. »PRINT 2/0«).

3/BEREICHSUEBERSCHREITUNG

Der Zahlenbereich von Real wird bei einer direkten Berechnung überschritten (z. B. »PRINT 10 ↑ 50«).

4/FEHLER BEI DER BERECHNUNG

Es ist bei einer direkten Berechnung irgendein Rechenfehler aufgetreten, der nicht genauer bekannt ist.

5/ABBRUCH MIT RESTORE

Während der Kompilierung wurde <RESTORE> gedrückt.

7/CODE VERSCHOBEN

Der erzeugte Code der vorangegangenen Zeile in Pass 2 ist gegenüber dem in Pass 1 erzeugten Code kürzer oder länger. Dieser Fehler darf nur dann auftreten, wenn andere Fehler vorausgegangen sind (z. B. »Konstante nicht definiert«). Er verschwindet, wenn die anderen Fehler behoben sind. Wenn er allein auftritt, dann wurde einer Variablen zweimal der gleiche Wert zugeordnet.

8/FALSCHER IF-ZWEIG

Wenn dem Ausdruck nach IF weder GOTO noch THEN folgt, dann erscheint dieser Fehler. Es kann aber auch der Ausdruck nach IF in einer Art und Weise fehlerhaft sein, daß der Compiler das Ausdrucksende zu früh vermutet.

9/UNBEKANNTER TOKEN

Diese Meldung erscheint dann, wenn der Basic-Boss einen Token (ein codierter Basic-Befehl) findet, den er nicht kennt. Das ist meist bei der Kompilierung von Programmen der Fall, die mit Hilfe einer Basic-Erweiterung erstellt wurden und Befehle dieser Erweiterung enthalten.

10/ES IST LONGIF NOTWENDIG

Dieser Fehler kann nur im SHORTIF-Modus auftauchen und zeigt an, daß die IF-Zeile für diesen Modus zu viel Speicher benötigt. Abhilfe schafft der »£LONGIF«-Befehl entweder am Programmanfang oder kurz vor der bemängelten Zeile. Hinter dieser Zeile kann man wieder auf »£SHORTIF« umschalten (für eine automatische Umschaltung bräuchte man einen dritten Pass).

11/ZU VIELE PARAMETER

Hier vermutet der Basic-Boss einen Befehl mit zu vielen Parametern. Der Fehler kommt dann zustande, wenn der Basic-Boss am Befehlsende angekommen zu sein glaubt, dann aber weder einen Doppelpunkt noch ein Zeilenende findet. Eine typische Ursache sind z. B. an den SYS-Befehl angehängte Parameter. Dann ist der »£OTHERON«-Befehl zu empfehlen.

12/UNZULAESSIGE ZUWEISUNG

Es wurde z. B. versucht, einer Systemvariablen etwas zuzuweisen, obwohl dies nicht erlaubt ist. Dies ist z. B. bei TI der Fall. ST kann man dagegen etwas zuweisen.

13/UNBEKANNTER BEFEHL (=)

Der Compiler glaubt, eine Variablenzuweisung zu bearbeiten, findet aber hinter der vermuteten Variable kein Gleichheitszeichen. Dann liegt meist ein Befehl vor, der falsch geschrieben wurde und den der Editor des Interpreters darum nicht als Token erkannt hat. Es kann sich aber auch um einen langen Variablennamen handeln, der Basic-Befehlsworte enthält, was nicht erlaubt ist.

14/UNBEKANNTER ZUSATZBEFEHL

Der Compiler hat den Zusatzbefehl hinter dem Linkspfeil nicht erkannt. Vermutlich wurde er falsch geschrieben oder es handelt sich um den Befehl einer Basic-Erweiterung.

15/ADRESSBEFEHL ZU SPAET

Ein Befehl wie »£ROUTSTART«, »£PROGSTART« usw. steht nicht am Anfang des Programms, wie es eigentlich sein sollte und kann daher nicht verwertet werden.

16/COMPILERDIREKTIVE UNBEKANNT

Der Compiler hat eine mit einem Pfundzeichen eingeleitete Compilerdirektive nicht verstanden. Es liegt vermutlich ein Schreibfehler vor.

17/DATENTYP ERWARTET

Der Compiler hat an dieser Stelle einen Datentyp wie »Byte« oder »Word« erwartet. Vermutlich liegt ein Schreibfehler vor.

18/ZU LANGER NAME

Der angegebene Dateiname ist zu lang und wurde auf 20 Zeichen gekürzt. Dieser Fehler tritt auch bei »£SYSTEXT« auf.

19/DATEINAME ERWARTET

Der Compiler hat an dieser Stelle einen in Anführungszeichen gesetzten Dateinamen erwartet. Vermutlich fehlt das Anführungszeichen. Dieser Fehler kann auch bei »£SYSTEXT« auftreten.

20/ES WURDE BEREITS CODE ERZEUGT

Der Basic-Boss hat eine Direktive gefunden, die nicht am Anfang eines Programms steht, dort aber stehen sollte, da sie sonst nicht einwandfrei funktioniert (z. B. »£SETPREFERENCES«).

21/FOR OHNE NEXT (FASTFOR)

Der Fehler tritt nur im FASTFOR-Modus auf und dann auch nur am Ende des Programms oder beim »£SLOWFOR«-Befehl. In beiden Fällen müssen nämlich sämtliche FOR-Schleifen abgeschlossen sein. Sind sie dies nicht, so erfolgt dieser Fehler. Man sollte also nach einem FOR-Befehl suchen, für den noch kein NEXT-Befehl existiert.

22/GO OHNE TO

Statt »GOTO« kann man auch »GO TO« schreiben. Wenn man jedoch »GO« ohne »TO« schreibt, beschwert sich der Basic-Boss.

23/UNBEKANNTE ZEILE

Es wurde eine Zeilennummer verwendet, für die keine Zeile existiert. Man sollte also untersuchen, wohin man eigentlich springen will, um dann entweder die vorangehende oder die nachfolgende Zeile anzusprechen.

24/' , ' ODER ')' ERWARTET

Beim DIM-Befehl wird hinter jeder Dimensionsangabe entweder ein Komma oder eine schließende Klammer erwartet. Vermutlich wurde das Ausdrucksende zu früh vermutet, wofür irgendein Schreibfehler verantwortlich sein dürfte.

25/FALSCHE DIMENSIONSANZAHL

Beim DIM-Befehl wurde ein Array dimensioniert, das bereits im vorangegangenen Programmtext verwendet wurde. Dabei müssen jeweils gleich viele Dimensionen benutzt werden. Wenn dies nicht beachtet wird, dann erscheint dieser Fehler.

26/'=' ERWARTET

Hinter der bei FOR angegebenen Schleifenvariablen muß ein »=« stehen, das der Basic-Boss hier nicht finden kann. Möglich ist auch ein langer Variablenname, in dem sich Basic-Befehle befinden, was nicht erlaubt ist.

27/'TO' ERWARTET

Beim FOR-Befehl wurde an der Stelle von »TO« etwas anderes gefunden. Möglich ist ein Schreibfehler oder ein Fehler in der Zuweisung hinter »FOR«, so daß dessen Ende zu früh vermutet wird.

28/NEXT OHNE FOR (FASTFOR)

Es wurde im FASTFOR-Modus versucht, eine Schleife mit NEXT abzuschließen, die es gar nicht gibt. Ursache hierfür könnte z. B. sein, daß der »£FASTFOR«-Befehl im Programm hinter dem angepeilten FOR-Befehl steht statt davor. In jedem Fall sollte man das FOR-NEXT-Gefüge untersuchen.

30/UNERLAUBTER ZAEHLERTYP

Bei FOR dürfen als Zählervariable nur numerische Variablen verwendet werden und keine String-Variablen. Auch Arrays sind verboten.

31/KOMMA ERWARTET

Der Basic-Boss erwartet an dieser Stelle ein Komma. Es kann allerdings auch sein, daß das Ende des vorangegangenen Ausdrucks zu früh vermutet wird. Man sollte hier im Zweifelsfall den gesamten Ausdruck überprüfen

32/NACH DEM INPUTSTRING MUSS ';' FOLGEN

Wenn nach dem INPUT-Befehl ein in Anführungszeichen geklammerter String steht, muß nach diesem String ein »;« folgen. Danach folgt die einzulesende Variable.

33/NACH ON MUSS GOSUB ODER GOTO FOLGEN

Nach dem ON-Befehl folgt ein Ausdruck, dem wiederum GOTO oder GOSUB folgen. Wenn diese Reihenfolge nicht eingehalten wird, erfolgt dieser Fehler. Möglich ist auch ein Fehler im Ausdruck hinter ON, der dazu führt, daß das Ausdrucksende zu früh vermutet wurde.

34/DAS ARRAY WURDE BEREITS DIMENSIONIERT

Es wurde versucht, ein Array zweimal mit DIM zu dimensionieren. Man sollte die andere DIM-Anweisung suchen und eliminieren.

35/FALSCHE NEXT-VARIABLE (FASTFOR)

Dieser Fehler tritt nur im FASTFOR-Modus auf. Es wird versucht, eine FOR-Schleife mit »NEXT X« abzuschließen, wobei aber die bei NEXT angegebene Variable nicht mit der bei FOR angegebenen identisch ist. Entweder wurde die Variable falsch geschrieben oder das FOR-NEXT-Gefüge ist durcheinander.

36/BEI DER FELDDIMENSIONIERUNG DUERFEN KEINE VARIABLEN VERWANDT WERDEN

Wenn Sie ein Array mit »DIM A(B)« dimensionieren, dann erhalten Sie diesen Fehler, weil der Basic-Boss keine variable Dimensionierung verarbeiten kann. Man muß also konstante Zahlen verwenden wie z. B. »DIM A(1000)«. Falls die Feldgröße vor dem Programmstart nicht bekannt ist, muß man das Feld wohl oder übel auf die größtmögliche Größe dimensionieren und dabei Speicherplatzverluste in Kauf nehmen. (Siehe auch unter »Änderungen beim Commodore-Basic V2« in Kapitel 2.6.4 »Problemfälle«.)

38/NUMMER ERWARTET

Es wurde eine Zahl zwischen 0 und 65535 erwartet. Es darf an dieser Stelle keine Variable verwendet werden.

39/KLAMMER ZU FEHLT

Es fehlt eine schließende Klammer im Ausdruck, oder das Ausdrucksende wird zu früh vermutet. Möglich ist auch eine überzählige öffnende Klammer.

40/ZAHL ERWARTET

Es wurde eine Zahl erwartet. Ein String, eine Variable oder ein arithmetischer Ausdruck an dieser Stelle lösen den Fehler aus. Dieser Fehler kann z. B. bei DATA auftauchen, wenn der Datentyp numerisch gewählt wurde (z. B. mit »£DATATYPE Byte«).

41/KLAMMER AUF FEHLT

Es fehlt eine öffnende Klammer im Ausdruck, oder eine schließende Klammer ist überzählig.

42/ZU VIELE FUNKTIONSPARAMETER

Es wurde versucht, einer Funktion zu viele Parameter mitzugeben. Möglich ist auch eine fehlende schließende Funktionsklammer bei verschachtelten Funktionen und Arrays.

43/DAS KOMMA MUSS INNERHALB EINER FUNKTIONS/ARRAY-KLAMMERUNG STEHEN

Der Basic-Boss kann mit einem im Ausdruck auftretenden Komma nichts anfangen. Dieser Fehler kann nur innerhalb einer Klammerung auftreten.

44/STRING HIER NICHT ERLAUBT

Der Fehler wird ausgelöst, wenn in einer Funktion ein Stringausdruck verwendet wird, obwohl ein numerischer Ausdruck verlangt ist (z. B. »PRINT ABS(A$)«).

45/ZUWENIG FUNKTIONSPARAMETER

Einer Funktion wurden zuwenig Parameter mitgegeben. Denkbar ist auch eine überflüssige schließende Klammer.

46/ZUVIEL ARRAYPARAMETER

Es wurde versucht, bei einem Array mehr Dimensionen anzusprechen, als dieses laut seiner Definition mit DIM besitzt (die erstmalige Verwendung eines Arrays betrachtet der Basic-Boss auch als Definition). Möglich ist aber auch eine fehlende schließende Klammer bei verschachtelten Funktionen und Arrays.

47/ZUWENIG ARRAYPARAMETER

Es wurden bei der Verwendung von Arrays weniger Dimensionen angegeben, als diese laut ihrer Definition mit DIM besitzen (die erstmalige Verwendung eines Arrays betrachtet der Basic-Boss auch als Definition). Möglich ist aber auch eine überzählige schließende Klammer bei verschachtelten Funktionen und Arrays.

48/DAS ARRAY MUSS MINDESTENS EINDIMENSIONAL SEIN

Wenn bei der erstmaligen Benutzung eines Arrays weniger als ein Parameter angegeben wird, dann erscheint dieser Fehler.

49/UNERWARTETES AUSDRUCKENDE

Wenn ein Ausdruck unvermittelt abbricht, dann erhalten Sie diese Meldung. Der Basic-Boss erwartete hier möglicherweise noch eine Zahl.

50/DEKLARATION ZU SPAET

Die Deklaration einer Variablen mittels einer Compilerdirektive (z.B. »£WORD A,B«) erfolgte erst, nachdem bereits der Code erzeugt wurde. Darum kann die Deklaration dieser Variablen nicht berücksichtigt werden. Sie sollte deshalb möglichst am Anfang des Programms stehen.

51/EIN STRING KANN NICHT IN EINE ZAHL UMGEWANDELT WERDEN

Es wurde ein Stringausdruck an einer Stelle benutzt, an der ein numerischer Ausdruck hätte stehen sollen.

52/EINE ZAHL KANN NICHT IN EINEN STRING UMGEWANDELT WERDEN

Es wurde ein numerischer Ausdruck an einer Stelle benutzt, an der ein Stringausdruck hätte stehen sollen. Möglicherweise wurde versucht, einen numerischen Ausdruck mit einem String zu verknüpfen.

53/FUNKTIONSPARAMETER MUESSEN IN KLAMMERN EINGESCHLOSSEN SEIN

Hinter einer Funktion fehlt die öffnende Klammer, die dort unbedingt stehen muß (»A= SIN 10« ist nicht möglich). Man kann auf die Umklammerung also grundsätzlich nicht verzichten.

54/DEKLARATION EINER SYSTEMVARIABLEN

Es wurde versucht, eine Systemvariable (wie TI, TI$, ST) zu deklarieren, was aber sinnlos ist, weil diese bereits von vornherein deklariert sind.

55/VARIABLE ERWARTET

An dieser Stelle wurde eine Variable oder gegebenenfalls auch eine Zahl erwartet, aber nicht gefunden.

57/DER SPEICHER IST ZU KNAPP

Dem Basic-Boss ist sein interner Speicher für Variablen, Zeilennummern, Konstanten usw. ausgegangen. Dies dürfte recht selten der Fall sein, da der Basic-Boss ab Version 2.2 sehr sparsam mit seinem Hauptspeicher umgeht.

58/DIMENSION UEBERSCHRITTEN

Bei der Verwendung von Konstanten als Parameter bei Array-Variablen kann der Basic-Boss sofort feststellen, ob diese Parameter den dimensionierten Bereich überschreiten. Wenn also z. B. »DIM A(15)« im Programm steht und an anderer Stelle »PRINT A(20)« verwendet wird, dann erscheint dieser Fehler.

60/DISKFEHLER:...

Das Floppy-Laufwerk hat dem Compiler einen Fehler signalisiert. Hinter »DISKFEHLER:« erhalten Sie dann die Meldung, die aus der Floppy ausgelesen wurde. Deren Bedeutung können Sie im Floppy-Handbuch nachschlagen. Nachdem ein Diskfehler aufgetreten ist, wird der Kompiliervorgang abgebrochen.

61/'FAST' ERWARTET

Wenn bei der Deklaration hinter einer Variablen ein Gleichheitszeichen folgt, dann muß diesem entweder eine Zahl oder »FAST« folgen. Ansonsten erscheint dieser Fehler (siehe unter »Datentypen« bei 2.3.2 »Die Deklaration«).

62/ZUVIEL 'FAST'-VARIABLEN

Es können maximal vier Byte mit FAST-Variablen belegt werden. Dies entspricht zwei Word/Integer-Variablen oder vier Byte-Variablen. Wenn diese vier Byte verbraucht sind, dann erscheint dieser Fehler.

63/VERBOTENER DATENTYP

Es können nur Variablen vom Typ Byte, Word und Integer als FAST-Variablen deklariert werden. Diese können außerdem keine Arrays sein. Der Fehler kann auch auftreten, wenn versucht wird, einer String-Variablen eine Adresse zuzuweisen, was ebenfalls verboten ist. Möglicherweise sollte auch ein Array als Konstante deklariert werden.

64/FUNKTIONSDEFINITION FALSCH

Es wurde ein Fehler bei der Definition einer FN-Funktion mit »DEF« gemacht. Möglicherweise sollten Sie sich die Syntax der Funktionsdefinition noch einmal im Basic-Handbuch ansehen.

65/FUNKTION MEHRFACH DEFINIERT

Dieselbe FN-Funktion wurde mehrfach im Programm mit »DEF« definiert. Der Interpreter läßt dies zu, der Basic-Boss nicht.

66/FUNKTION NICHT DEFINIERT

Es wurde versucht, eine FN-Funktion anzuwenden, die zu diesem Zeitpunkt noch nicht mit »DEF« definiert war. Eine solche Funktion muß in einer kleineren Zeilennummer und vor ihrer Anwendung definiert sein.

67/VERBOTENER TYP BEI FN

Wenn eine FN-Funktion gefunden wurde, deren Parameter ein String ist, dann erscheint diese Meldung (z.B. bei »FN("abc")«.

68/KONSTANTE BEREITS DEFINIERT

Es wurde versucht, einer Variablen vom Typ Constant zweimal einen Wert zuzuweisen. Das ist jedoch nur ein einziges Mal im gesamten Programm erlaubt.

69/KONSTANTE NICHT DEFINIERT

Es wurde eine Variable des Datentyps Constant benutzt, obwohl sie noch nicht in einer vorausgehenden Zeile definiert wurde. Siehe hierzu unter »Datentypen« Kapitel 2.3.10 »Der Datentyp Constant«.

70/FUNKTION/KONSTANTE ERWARTET

Es wird versucht, eine FN-Funktion oder eine Konstante zu definieren, wobei allerdings der Funktionstext bzw. der Konstantentext fehlt. Es liegt also ein schwerer syntaktischer Fehler bei »DEF FN« oder der Konstantenzuweisung vor.

71/COMPILERFUNKTION UNBEKANNT

Der Compiler hat eine mit »£« eingeleitete Funktion gefunden, die er nicht kennt. Meist ist ein Schreibfehler die Ursache. Siehe auch Kapitel 2.5 »Neue Befehle und Funktionen«.

2.12.2 Die Meldungen des Compilats

Dies sind die Fehlermeldungen, die während der Ausführung des kompilierten Programms ausgegeben werden. Die Nummer der betroffenen Zeile wird normalerweise nicht angezeigt. Dieses Manko kann man beheben, wenn man »£LINEON« benutzt (siehe unter »£LINEON«in Kapitel 2.4.8 »Sonstige Direktiven«. Die Fehler sind nicht numeriert, was nicht weiter schlimm ist, da es nicht sehr viele gibt, denn die meisten Fehler werden bereits vom Compiler abgefangen. Es werden die normalen englischen Fehlermeldungen des Interpreters benutzt. Sie haben meist auch dieselbe Bedeutung wie beim Interpreter.

Die Fehlerliste

NEXT WITHOUT FOR ERROR

Das Programm ist bei einem NEXT-Befehl angelangt, ohne daß zuvor ein FOR-Befehl abgearbeitet wurde. Man sollte den Programmlauf also genau überprüfen, um festzustellen, wann dieses NEXT angesprungen wird. Dieser Fehler kann bei einer FASTFOR-Schleife nicht auftreten.

OUT OF DATA ERROR

Dem READ-Befehl sind die Daten ausgegangen. Man sollte also überprüfen, wann und wo dieser READ-Befehl ausgeführt wird und auf welche Daten er normalerweise zugreifen sollte.

TYPE MISMATCH ERROR

Dieser Fehler hat beim Compilat normalerweise eine andere Bedeutung als beim Interpreter. Er zeigt an, daß die einzulesenden Daten beim READ-Befehl von einem anderen Typ sind als die Variable, in die sie eingelesen werden sollen. Wenn man z. B. alle Daten mit »£DATATYPE Byte« als Byte-Daten erklärt hat und später der Befehl »READ R« ausgeführt wird (R sei eine Fließkommavariable), dann erscheint dieser Fehler. Wenn man die Daten trotzdem in eine Fließkommavariable einlesen will, muß man sie zunächst in eine Byte-Variable einlesen, um sie anschließend einer Real-Variablen zuzuweisen (z. B. »READ B: R=B«, wobei B eine Byte-Variable sein soll). Wenn die Daten vom Typ String sind, tritt dieser Fehler normalerweise nicht auf, da der READ-Befehl Strings in jeden anderen Typ umwandeln kann. Er erscheint aber trotzdem, wenn die Umwandlung unmöglich ist (man kann keinen Buchstabenstring in eine Zahl umwandeln).

BAD SUBSCRIPT ERROR

Es wurde versucht, ein Array-Element anzusprechen, das nicht existiert. Die Feldgrenzen wurden also überschritten. Wenn man z. B. mit »DIM A(100)« ein Feld dimensioniert und dieses anschließend mit »I=101: A(I)=1« anspricht, dann erhält man diesen Fehler. Standardmäßig tritt dieser Fehler aber nicht bei allen Feldern auf. Bei eindimensionalen Word-, Integer- oder Byte-Feldern wird eine Feldüberschreitung nicht bemerkt, es sei denn, man benutzt »SLOWARRAY«.

OUT OF MEMORY ERROR

Wenn der Stringspeicher nicht mehr ausreicht, dann erscheint dieser Fehler. Man muß dann entweder den Stringspeicher vergrößern (mit »HEAPSTART« und »HEAPEND« oder mit »ALLRAM«, siehe Kapitel 2.4 »Beschreibung der Direktiven«) oder mit den Strings etwas sparsamer umgehen.

CAN'T CONTINUE ERROR

Der Fehler hat nichts mehr mit seiner ursprünglichen Bedeutung gemein (er kann ja normalerweise nur im Direktmodus auftauchen). Das Compilat (genauer: die Garbage-collection) zeigt mit ihm an, daß der Stringspeicher defekt ist. Grund hierfür kann entweder ein unvorsichtiger POKE in den Stringspeicher sein oder schlimmstenfalls ein Fehler in der Stringverwaltung des Compilers.

STRING TOO LONG ERROR

Der Fehler kann nur beim Addieren von Strings auftreten (z.B. »A$=B$+C$«). Die zu addierenden Strings haben eine Gesamtlänge von mehr als 255 Zeichen, was auch meine Stringverwaltung nicht verarbeitet.

ILLEGAL QUANTITY ERROR

Dieser Fehler kann bei der MID$-Funktion ausgelöst werden, wenn der zweite Parameter »0« ist (z.B. »MID$ (A$, 0, 2)«). Er muß mindestens »1« sein.

SYNTAX ERROR

Wenn mit »£OTHERON« erweiterte SYS-Befehle erlaubt sind, dann kann auch dieser Fehler auftreten. Er wird vom Compilat ausgelöst, wenn die Parameterübergabe an das aufgerufene Maschinenprogramm durcheinandergekommen ist. Bei »£OTHERON« sind weitere Fehler denkbar, die das aufgerufene Maschinenprogramm auslöst.
Es sind auch noch andere Meldungen möglich, die vom Basic-Interpreter des C64/C128 stammen. Diese Meldungen haben im allgemeinen dieselbe Bedeutung wie sonst auch (z. B. »division by zero error« oder »overflow error«). Sie können auftreten, weil die arithmetischen Routinen des Basic-ROM vom Compilat teilweise genutzt werden (für Realzahlen).

Anhang A Übersicht aller Direktiven

£ALLRAM	Schaltet die Nutzung des Gesamtspeichers (62 Kbyte) ein
£BASICRAM	Schaltet die Nutzung des Gesamtspeichers (62 Kbyte) aus
£BOOLEAN	Legt den Variablentyp BOOLEAN (zwei Zustände) fest
£BOOLEANTYPE	Bestimmt den Erwartungstyp bei IF-THEN
£BOSSOFF	Schaltet Basic-Boss-Direktiven aus
£BOSSON	Schaltet Basic-Boss-Direktiven ein
£BYTE	Legt den Variablentyp Byte (256 Zustände) fest
£CALCTYPE	Bestimmt den Mindesttyp aller Berechnungen
£CODE	Fügt Codes in das Compilat ein
£DATAFILE	Bestimmt Gerät und Name des Daten-Files
£DATASTART	Legt die Startadresse des Datencodeteils fest
£DATATYPE	Legt den Typ der DATA-Daten fest
£FASTARRAY	Schaltet die schnellen, aber gefährlichen Feldzugriffe ein
£FASTFOR	Schaltet die schnelle FOR-NEXT-Schleife ein
£HEAPEND	Bestimmt die Endadresse des Stringspeichers
£HEAPSTART	Bestimmt die Anfangsadresse des Stringspeichers
£HELPSTART	Bestimmt die Anfangsadresse des Hilfespeichers
£IGNORE	Ignoriert alle nachfolgenden Befehle
£INITOFF	Schaltet die Initialisierung der Variablen auf »0« aus
£INITON	Schaltet die Initialisierung der Variablen auf »0« ein
£INTEGER	Legt als Variablentyp »Integer« fest
£LINEOFF	Zeilenaktualisierung ausschalten
£LINEON	Zeilenaktualisierung einschalten
£LONGIF	Normale IF-Anweisung verwenden
£LONGNAME	Schaltet auf Basic-Boss-Variablen-Erkennung
£OTHER	Erlaubt unbekannte Befehle aus Basic-Erweiterungen

£OTHEROFF	Verbietet zusätzliche Parameter des SYS-Befehls
£OTHERON	Erlaubt zusätzliche Parameter des SYS-Befehls
£PRINTTYPE	Legt den Mindest-Variablen-Typ bei PRINT-Anweisungen fest
£PROGFILE	Legt Gerät und Name der Programm-Datei fest
£PROGSTART	Legt die Anfangsadresse des Programms fest
£PROTOCOL	Veranlaßt die Ausgabe von Fehlermeldungen und sonstigen Daten an den Drucker oder in eine Datei
£RAM	Schaltet im ALLRAM-Modus den schnellen RAM-Zugriff für POKE und PEEK ein
£REAL	Legt als Variablentyp »Real« fest
£ROM	Schaltet im ALLRAM-Modus den schnellen RAM-Zugriff für POKE und PEEK aus
£ROUTFILE	Bestimmt Gerät und Name der Routinen-Datei
£ROUTSTART	Legt Startadresse des Routinen-Teils fest
£SETPREFERENCES	Erhebt die momentanen Einstellungen zu den dauerhaften Standardeinstellungen
£SHORTIF	Schnelle IF-Anweisung verwenden
£SHORTNAME	Schaltet auf zweistellige Variablenerkennung
£SLOWARRAY	Schaltet auf normale Feldzugriffe zurück
£SLOWFOR	Normale FOR-NEXT-Schleife verwenden
£SOURCEFILE	Legt Name, Gerät und Sekundäradresse der Quelldatei fest
£STRING	Legt als Variablentyp »String« fest
£SYSNUMBER	Regelt die Zeilennummer der SYS-Zeile
£SYSOFF	Schaltet die SYS-Zeile des Compilats aus
£SYSON	Schaltet die SYS-Zeile des Compilats ein
£SYSTEXT	Legt den Text in der SYS-Zeile fest
£USE	Deaktivierung von »£IGNORE«
£VARSTART	Legt Startadresse des Variablenspeichers fest
£WORD	Legt als Variablentyp »Word« fest

Anhang B — Zusätzliche Befehle und Funktionen

←BYTE	Schreibt ein Byte in den aktuellen Ausgabekanal
←CLI	Erlaubt Interrupts
←IN	Legt den Eingabekanal in die angegebene Datei
←INTEGER	Schreibt zwei Byte in den aktuellen Ausgabekanal
←IOROM	Schaltet I/O-Bereich und Basic-ROM ab
←LOAD	Lädt eine Datei in den Speicher und arbeitet den nächsten Befehl ab
←OUT	Legt den Ausgabekanal in die angegebene Datei
←RAM	Schaltet das ROM ab
←REAL	Schreibt fünf Byte in den aktuellen Ausgabekanal
←RESET	Legt den Ausgabekanal auf den Bildschirm und den Eingabekanal auf die Tastatur
←ROM	Schaltet das ROM ein
←SEI	Verbietet Interrupts
←WORD	Schreibt zwei Byte in den aktuellen Ausgabekanal
@BYTE	Liest ein Byte aus dem aktuellen Ausgabekanal
@INTEGER	Liest zwei Byte aus dem aktuellen Ausgabekanal
@REAL	Liest fünf Byte aus dem aktuellen Ausgabekanal
@WORD	Liest zwei Byte aus dem aktuellen Ausgabekanal

Anhang C Automatikmodus

Bedeutung der Speicheradressen:

1000,1001	Enthalten sie die Werte »123« und »234«, so wird der Automatikmodus aktiviert
1002	Gerätenummer Quelldatei
1003	Sekundäradresse
1004	Länge Dateiname
1005–1020	Filename
900	Gerätenummer Zieldatei
901	Sekundäradresse
902	Länge Dateiname
903–918	Filename

Anhang D Wertebereiche der Variablentypen

Typ	Wertebereich Minimum	Wertebereich Maximum	Speicherplatz
Real	+–2.93873588 * 10^{-39} bis	+–1.70141183 * 10^{38}	5 Byte
Integer	–32768 bis	+32767	2 Byte
Word	0 bis	65535	2 Byte
Byte	0 bis	255	1 Byte
Boolean	0 und	–1	1 Byte

Anhang E — Der Software-Speeder Ultraload Plus

Das Laden des Basic-Boss dauert ohne Floppy-Speeder zirka zwei Minuten. Sollten Sie mehrere Programme kompilieren, so ergibt sich eine beträchtliche Ladezeit. Mit dem hier beschriebenen Programm »Ultraload Plus« können Sie die Ladezeit um ein Vielfaches verkürzen. Der Basic-Boss, Ihre Basic-Programme und Ihre Compilate werden dann in siebenfacher Geschwindigkeit geladen. Bitte laden Sie das Programm mit:

```
load "ultraload plus",8
run
```

Jetzt steht Ihnen die erhöhte Ladegeschwindigkeit zur Verfügung. Ihre Programme laden Sie wie gewohnt. Die nachfolgende Beschreibung von »Ultraload Plus« ist in zwei Teile gegliedert: Der erste Teil beschreibt das Programm für den Anwender (die Vorteile und Möglichkeiten), während sich der zweite Teil an interessierte Programmierer wendet, die mehr über dieses System wissen wollen. Doch zuerst zu einer einfachen Beschreibung dieses Programms. Auch wenn Sie Ihre Floppy für den C64 (C128) vielleicht noch nicht lange haben, so werden Sie sicher schon die berüchtigt langen Wartezeiten der 1541/1570/1571 kennengelernt haben. Es gibt nun mehrere Lösungen, die langsame Geschwindigkeit der Diskettenstation zu umgehen, sei es hardware- oder softwaremäßig. Ultraload Plus ist eine der Softwarelösungen und noch dazu eine sehr gute. Im folgenden sind die Vorteile für Sie als Benutzer aufgeführt:

- Programme werden 6,8mal schneller geladen als normal.
- Directory-Einträge werden sehr viel schneller gefunden.
- Auch der Befehl »VERIFY« wird sehr viel schneller ausgeführt.
- Da der Schreib-Lese-Kopf der Floppy mehr als dreimal so schnell bewegt wird wie normal, wird somit einer Dejustierung vorgebeugt.
- Sie können das Laden jederzeit mit <RESTORE> unterbrechen.

Vielleicht kennen Sie bereits das Programm »Hypra-Load«. Aber Ultraload Plus ist auch noch besser als Hypra-Load, wie die folgende Zusammenstellung beweist.

- Das Laden erfolgt noch schneller als bei Hypra-Load (6,8mal anstelle von 6,2mal).
- Hypra-Load beschleunigt nicht die Suche von Directory-Einträgen.
- Die Wahrscheinlichkeit eines Absturzes ist bei Ultraload Plus noch geringer als bei Hypra-Load.
- Mit Hilfe mehrerer Parameter läßt sich Ultraload Plus einfacher an Programme anpassen, die zuerst nicht mit Floppy-Speedern (die Bezeichnung für solche Beschleunigungsprogramme) zusammenarbeiteten.
- Ultraload Plus gibt bei einem Schreib-Lese-Fehler eine dementsprechende Meldung aus, ohne – wie Hypra-Load – abzustürzen.
- Bei Ultraload Plus wird der Bildschirm nicht abgeschaltet.

Sie sehen also, daß Ultraload Plus einige Vorteile mit sich bringt. Vor allem werden Ihnen aber die wesentlich kürzeren Ladezeiten auffallen. Vorbei sind die Kaffeepausen beim Laden des 193 Block langen Basic-Boss. Um mit Ultraload Plus zusammenzuarbeiten, brauchen Sie es nur mit den oben genannten Befehlen zu starten. Danach stehen Ihnen die schnelleren Laderoutinen sofort zur Verfügung. In der letzten Zeile der Einschaltmeldung sehen Sie einen SYS-Befehl (SYS 336). Sie können ihn verwenden, sollten Sie <RUN-STOP/RESTORE> gedrückt oder einen Reset ausgelöst haben. Auch dann können Sie die schnelleren Laderoutinen sofort weiterverwenden, ohne das Programm neu laden zu müssen.
Sollte es ein Programm geben, das nicht problemlos mit dem Floppy-Speeder läuft, möchte ich Sie auf den folgenden Teil verweisen; lassen Sie sich nicht durch etwaige Spezialausdrücke abschrecken. Zwischendurch werden Sie immer wieder wertvolle Hinweise für die Arbeit mit diesem Programm finden, wie z. B. die Steigerung der Geschwindigkeit auf das bis zu 8fache! Des weiteren werden noch zwei Hilfsmittel (Tools) mitgeliefert, die auch im folgenden beschrieben werden. Die schnelle Datenübertragung vom Diskettenlaufwerk zum Computer beruht auf der gleichzeitigen Übertragung von zwei Bit, auf dem eingeschränkten Handshake-Betrieb sowie auf der zeitsparenden Verwendung einer Tabelle. Bei der Suche nach den Directory-Einträgen wird eine eigene GCR-Codierung verwendet (Group-Code-Recording, so heißt das Aufzeichnungsverfahren der 1541). Die rasante Datenübertragung zum Laufwerk ist

für das um Sekundenbruchteile verzögerte Starten des Motors zuständig und beruht auf einer äußerst kurzen Transfer-Routine. Das Laden des Directorys (LOAD "$",8 ...) wird nicht beschleunigt. Dafür wird der VERIFY-Befehl – wie oben bereits erwähnt – schneller abgearbeitet. Im Gegensatz zur Original-Verify-Routine bricht das Programm beim Auftreten eines Fehlers sofort ab. Mit der Befehlsfolge »PRINT PEEK(174) + 256 * PEEK(175)« können Sie das erste unterschiedliche Byte feststellen.
Wenn Sie das Programm direkt nach dem Laden listen, sehen Sie folgende Zeile:

```
1985 sys 2080,00288,192,214,n,3
```

Im folgenden werden wir die einzelnen Parameter näher erläutern. Dazu werden wir sie aber zuerst durch Buchstaben ersetzen, um sie einfacher beschreiben zu können. Demnach ist die Startzeile folgendermaßen aufgebaut:

```
1985 sys 2080, a, b, c, d, e
```

A: Startadresse des Bootprogramms
B: Highbyte der Anfangsadresse des Arbeitsbereichs; muß bei der zweiteiligen Version gleich der nachfolgenden Zahl sein und zwischen 16 und 200 liegen
C: Highbyte der Anfangsadresse des Hauptteils; muß bei der zweiteiligen Version zwischen 16 und 200, bei der dreiteiligen zwischen 16 und 246 liegen
D: Transfergeschwindigkeit (N für normal, H für hoch)
E: Speicherbelegungsart (2 für zweiteilig, 3 für dreiteilig)

Prinzipiell muß die Länge der Basic-Zeile immer gleich sein. Daher müssen eventuelle Veränderungen mit führenden Nullen versehen werden (beispielsweise 00288 statt 288).
Kommen wir jetzt aber endlich zu den Parametern: Ultraload Plus kann entweder als zweiteilige oder dreiteilige Version im Speicher liegen. Für jeden Teil existiert genau eine Startadresse (A–C). Da bei zwei Teilen die Adresse für den dritten Programmblock entfällt, muß in dem Fall die Angabe B mit Angabe C übereinstimmen (E muß 2 sein).

A wird einfach als Dezimalzahl angegeben, kann also Werte zwischen 0 und 65535 annehmen (das sind alle im C 64 möglichen Adressen). Hier empfiehlt es sich aber, bei der Adresse 288 zu bleiben. Die folgenden Ausführungen über die Angaben B und C werden nur Assemblerprogrammierer unter Ihnen verstehen. Das ist aber kein Grund zur Verzweiflung, da weiter unten noch eine Lösung für alle anderen Leser geboten wird. Die Bootroutine (ab 288) ruft das eigentliche Hauptprogramm (ca. 2 Kbyte) auf. Die Angaben für C (Hauptteil) ermitteln Sie, indem Sie jeweils die Highbytes der Adressen angeben (dezimal). Das heißt also, daß die Zahl 88 für die Startadresse 22528 (=$5800) steht. Wie bereits erwähnt, müssen bei der zweiteiligen Version B und C unbedingt übereinstimmen.
Dieser zweite Teil enthält alle Laderoutinen und einen kleinen Arbeitsspeicher. Er ist vor Überschreiben durch sich selbst geschützt, das heißt, er kann beim Laden nicht zerstört werden. Diesen Programmteil können Sie beispielsweise in das RAM unter dem Kernel oder in das RAM unter den I/O-Bausteinen legen. Der dritte und letzte Teil ist der sogenannte Arbeitsbereich (wird durch B festgelegt!). Auch dieser Bereich ist vor Überschreiben geschützt. Zunächst möchte ich dieses Prinzip der Speichereinteilung an einem Beispiel verdeutlichen:

```
1985 sys 2080, 00288, 192, 214, n, 3
```

Der Bootteil liegt demzufolge ab Adresse 288, der Arbeitsbereich ab 49152 (=$C000), der Hauptteil liegt ab 54784 (=$D600), und die »3« am Ende verdeutlicht noch einmal, daß es sich um eine dreiteilige Version handelt. Der fünfte Parameter gibt die Transfergeschwindigkeit an, mit der geladen und gespeichert werden soll. Dabei steht »N« für normal und »H« für hoch. Möchten Sie die Ladegeschwindigkeit um den Faktor 8 erhöhen, geben Sie bitte die folgenden Zeilen ein:

```
open 1,8,15
print #1, "m-w"+chr$(105)+chr$(0)+chr$(1)+chr$(7);
close1
```

Mit dieser kurzen Befehlssequenz wird der Blockabstand auf Diskette von normalerweise zehn auf sieben umgestellt. Dadurch kann Ultraload Plus noch schneller auf die Diskette zugreifen. Alle Programme, die so gespeichert wurden, werden später auch wieder schneller geladen. Allerdings funktioniert dieser Mechanismus nur, wenn Sie den Übertragungsmodus auf »H«

gestellt haben. Zum Schluß möchte ich noch auf den Fall eingehen, daß ein Programm nicht mit Ultraload zusammenarbeitet. Laden Sie dazu das Programm »ULTRALOAD TOOL 1«, und starten Sie es. Es füllt den Speicher mit einem bestimmten Code und führt danach einen Reset aus. Laden Sie daraufhin Ihr »Problemkind«, und lösen Sie wieder einen Reset aus (SYS 64738 oder notfalls mit einem Resetschalter). Laden Sie jetzt noch »ULTRALOAD TOOL 2«, und starten Sie auch dieses Hilfsmittel. Daraufhin werden Ihnen alle möglichen Startadressen für die einzelnen Programmteile von Ultraload Plus angezeigt. Diese Werte können Sie dann unmittelbar in den SYS-Befehl von Ultraload Plus übernehmen.

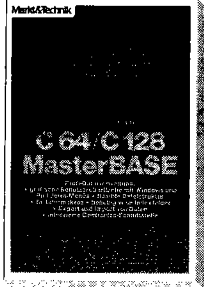
Anhang F Stichwortverzeichnis

(Thilo Herrmann/wb/ti)

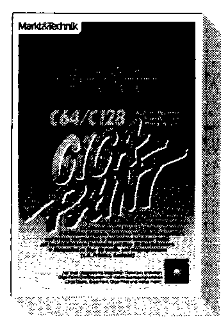
Bücher zum Commodore 64/128

S. Baloui
C64/C128 MasterBase-Dateiverwaltung
MasterBase ist eine Dateiverwaltung für den C64/C128 für gehobene Ansprüche.
1988, 155 Seiten, inkl. Diskette
Bestell-Nr. 90583
ISBN 3-89090-583-8
DM 59,-* (sFr 54,30*/öS 502,-*)

M. Pahl/T. Rullkötter/M. Kuk
C64/C128 MasterText Plus
MasterText 64, aus dem 64'er-Sonderheft 16, gehört zu den leistungsfähigsten 64er-Textprogrammen. In seiner verbesserten und erweiterten Plus-Version sind viele neue Hilfsprogramme wie »Master-Spiel« und »MasterAdress« sowie eine komfortable Druckeranpassung enthalten.
1988, 201 Seiten, inkl. Diskette
Bestell-Nr. 90527
ISBN 3-89090-527-7
DM 59,-* (sFr 54,30*/öS 502,-*)







W. Oppacher/K. Oppacher/
M. Wenzel
C64/C128 Giga-Paint
Giga-Paint, das Mal- und Zeichenprogramm der Superlative, unterstützt alle Grafikfähigkeiten Ihres C64 auf hervorragende Weise. Es hebt sich von der breiten Masse aller Grafikprogramme durch völlig neue Optionen hervor, wie z.B. die fantastischen 3-D- und Zufallsoperationen oder die Kompatibilität über 30 Malprogrammen.
1988, 261 Seiten, inkl. 2 Disketten
Bestell-Nr. 90619
ISBN 3-89090-619-2
DM 59,-* (sFr 54,30*/öS 502,-*)

S. Vilsmeier
3-D-Konstruktion mit Giga-CAD Plus auf dem C64/C128
Das 3-D-Konstruktionsprogramm Giga-CAD, das im 64'er-Sonderheft Grafik für Furore sorgte, ist eines der professionellsten CAD-Programme für den C64.
1986, 183 Seiten, inkl. 2 Disketten
Bestell-Nr. 90409
ISBN 3-89090-409-2
DM 49,-* (sFr 45,10*/öS 417,-*)

* Unverbindliche Preisempfehlung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.



Markt&Technik
Zeitschriften · Bücher
Software · Schulung

812/131-2

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Computerliteratur und Software vom Spezialisten

Vom Einsteigerbuch für den Heim- oder Personalcomputer-Neuling über professionelle Programmierhandbücher bis hin zum Elektronikbuch bieten wir Ihnen interessante und topaktuelle Titel für

• Apple-Computer • Atari-Computer • Commodore 64/128/16/116/Plus 4 • Schneider-Computer • IBM-PC, XT und Kompatible

sowie zu den Fachbereichen Programmiersprachen • Betriebssysteme (CP/M, MS-DOS, Unix, Z80) • Textverarbeitung • Datenbanksysteme • Tabellenkalkulation • Integrierte Software • Mikroprozessoren • Schulungen.

Außerdem finden Sie professionelle Spitzen-Programme in unserem preiswerten Software-Angebot für Amiga, Atari ST, Commodore 128, 128D, 64, 16, für Schneider-Computer und für IBM-PCs und Kompatible!

Fordern Sie mit dem nebenstehenden Coupon unser neuestes Gesamtverzeichnis und unsere Programmservice-Übersichten an, mit hilfreichen Utilities, professionellen Anwendungen oder packenden Computerspielen!



Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bitte schneiden Sie diesen Coupon aus, und schicken Sie ihn in einem Kuvert an: Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar

Markt&Technik Verlag AG
– Unternehmensbereich Buchverlag –
Hans-Pinsel-Straße 2
D-8013 Haar bei München

Bitte schicken Sie mir:
□ Ihr neuestes Gesamtverzeichnis
□ Eine Übersicht Ihres Programmservice-Angebotes aus der Zeitschrift ____________

□ Außerdem interessiere ich mich für folgende/n Computer: ____________

Adresse:
Name ____________
Straße ____________
Ort ____________







BASIC-BOSS
Basic-Compiler

Enthalten:
Eine
$5^1/_4$"-Programm-
diskette

